ANALISIS KINERJA ALGORITME PERHITUNGAN *GEODESIC DISTANCE* PADA PERMUKAAN OBJEK *TRIANGULAR MESH*

TUGAS AKHIR

Disusun Oleh :

EDDY TJANDRA

NRP. 5101 100 030

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2005

ANALISIS KINERJA ALGORITME PERHITUNGAN *GEODESIC DISTANCE* PADA PERMUKAAN OBJEK *TRIANGULAR MESH*

TUGAS AKHIR

Diajukan untuk Memenuhi Sebagian Persyaratan

Memperoleh Gelar Sarjana Komputer

Pada

Jurusan Teknik Informatika

Fakultas Teknologi Informasi

Institut Teknologi Sepuluh Nopember

Surabaya

Mengetahui/Menyetujui,

Dosen Pembimbing I	Dosen Pembimbing II
(Rully Soelaiman, S.Kom, M.Kom)	(I Made Agus Setiawan, S.Kom)
NIP. 132 085 802	NIP.

SURABAYA

JULI 2005

ABSTRAK

Dewasa ini perhitungan jarak terdekat dan pembuatan lintasan pada permukaan merupakan permasalahan umum dalam Computational Geometry. Masalah ini muncul pada beberapa aplikasi seperti perencanaan gerakan robot, navigasi terrain, dan Sistem Informasi Geografik. Untuk itu diperlukan suatu metode yang efisien dan efektif untuk menghitung geodesic distance dan membuat geodesic path.

Dalam Tugas Akhir ini dilakukan penerapan algoritme perhitungan geodesic distance pada permukaan objek triangular mesh untuk dibuktikan tingkat keakuratan dan efisiensinya. Algoritme yang diterapkan adalah Fast Marching Method on Triangulated Domain (FMM on TD) yang berjalan dengan kompleksitas waktu O(n lg n), dimana n adalah jumlah titik pada permukaan. Inti dari algoritme ini adalah melakukan front propagation dari titik awal ke segala arah yang mungkin sampai diperoleh titik akhir. Setiap bergerak maju algoritme ini selalu menghitung nilai jarak suatu titik terhadap titik awal. Setelah proses perhitungan geodesic distance selesai, dilakukan proses pembuatan geodesic path. Inti dari proses ini adalah melakukan back propagation pada permukaan dari titik akhir sampai diperoleh titik awal.

Berdasarkan uji coba, tingkat keakuratan algoritme FMM on TD adalah lebih dari 95%. Keakuratan ini dipengaruhi oleh jumlah segitiga pembentuk permukaan. Semakin banyak segitiga semakin akurat geodesic distance yang dihasilkan, tetapi waktu yang dibutuhkan untuk melakukan proses perhitungan menjadi semakin lama.

Kata Kunci : *Computational Geometry, Geodesic Distance, Geodesic Path, Triangular Mesh, Fast Marching Method on Triangulated Domain, Front Propagation, Back Propagation*

KATA PENGANTAR

Penulis sungguh bersyukur kepada Tuhan Yesus Kristus yang telah memampukan dan menyertai penulis sehingga dapat mengerjakan dan menyelesaikan Tugas Akhir berjudul:

ANALISIS KINERJA ALGORITME PERHITUNGAN *GEODESIC DISTANCE* PADA PERMUKAAN OBJEK *TRIANGULAR MESH*

Tugas Akhir ini dibuat guna memenuhi persyaratan akademik dalam rangka ujian akhir bagi mahasiswa Strata 1 (S1) Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember Surabaya.

Dalam Tugas Akhir ini penulis mengetengahkan salah satu algoritme yang digunakan untuk menghitung *geodesic distance* pada permukaan Objek *triangular mesh* yaitu algoritme *Fast Marching Method on Triangulated Domain.*

Bagaimanapun juga Penulis telah berusaha sebaik-baiknya dalam menyusun Tugas Akhir ini, namun penulis mohon maaf apabila terdapat kekurangan, kesalahan maupun kelalaian yang telah penulis lakukan. Penulis mengharapkan adanya kritik dan saran yang membangun guna menambah manfaat serta mengurangi kesalahan dan kekurangan yang ada.

Pada akhirnya penulis berharap semoga laporan ini dapat memberikan manfaat bagi kita semua.

Surabaya, 7 Juli 2005

Eddy Tjandra

UCAPAN TERIMA KASIH

Pada kesempatan ini, penulis ingin mengucapkan terima kasih yang sebesar-besarnya atas bantuan dan dukungan yang tak ternilai kepada:

1. Bapak Rully Soelaiman, S.Kom, M.Kom. selaku Dosen Pembimbing I, yang selalu memberikan dorongan, bimbingan, petunjuk, dan pelajaran "khusus" yang tidak diberikan oleh dosen-dosen lain, juga atas semua bantuan yang diberikan kepada penulis.
2. Bapak I Made Agus Setiawan, S.Kom. *(a.k.a. Kadek)* selaku dosen pembimbing kedua, yang setia memberikan motivasi, semangat, serta ilmunya kepada penulis baik ketika masih kuliah bersama ataupun ketika sudah menjadi dosen.
3. Bapak Yudhi Purwananto S.Kom, M.Kom. selaku Ketua Jurusan atas kebijakan, dorongan, bantuan, dan yang terutama adalah pengalaman ketika penulis diajak ke Jakarta (Le Meridien Hotel, 6-7 Desember 2004)
4. Bapak Ir. Muchammad Husni, M.Kom. selaku dosen wali penulis selama 8 semester yang selalu memberi dorongan kepada penulis.
5. Bapak Ir. F.X. Arunanto, MSc selaku Kepala Sie Laboratorium Komputing dan seluruh Dosen yang lain: Ibu Nanik S., Ibu Esther H., Ibu Handayani T., Ibu Chastine F., Ibu Bilqis A., Ibu Diana P., Ibu Siti R., Ibu Umi L. Yuhana, Ibu Wiwik A., Bapak Supeno Djanali, Bapak Arif Djunaidy, Bapak Khakim G., Bapak Aris T., Bapak Febrilian S., Bapak Fajar B., Bapak Riyanarto S., Bapak Faisal J., Bapak Royyana, Bapak

Imam K., Bapak Wahyu S., Bapak Irfan S., Bapak Waskhito W., Bang Daniel O. atas didikan dan ajaran kepada penulis selama menyelesaikan studi di Teknik Informatika ITS.

6. Seluruh staf dan karyawan Jurusan Teknik Informatika-ITS, Mbak Davi (eks. Pengurus RBTC), Mbak Eva, Mas Yudi, Pak Sugeng, Pak Narno, Pak Mu'in, Mbak Fatin, Karmono, Bu Tuti, dan karyawan lain.
7. Agustina Putri, terima kasih untuk segalanya yang telah kau beri, kehadiranmu sungguh telah mewarnai hidupku. Juga terima kasih untuk keluarga dari Putri, Bapak Agustinus S, dan Ibu, serta Eri dan Adri.
8. Mama di rumah, yang telah banyak memberikan dukungan dan doa, juga yang sudah membesarkan anak-anaknya dengan penuh kesabaran. Semoga ini menjadi kado yang terindah di hari ulang tahunmu, Mama.
9. Keluarga yang lain di rumah, Papa dan Kakak.
10. Ce Sylvia TG, yang menjadi *big sister in Christ.* Terima kasih atas doa, bimbingan, *advice*, *support*, serta kesediaan untuk mendengarkan segala keluh kesahku ketika aku banyak mengalami permasalahan hidup, *I owe you a lot, Sis!*.
11. Teman-teman Komsel-ku: Ko Alex, Budi, Welly, Yosafat, Filip, Lulung, dan alm. Adhi Cahyono. Terima kasih atas persahabatan, kebersamaan, suka dan duka yang telah kita lalui bersama. *Keep Forever Friend in Christ, Bro!*
12. Pdt. Agustina Manik, S.Th., M.Th. (a.k.a. Kak Nina), selaku pembimbing dari Remaja hingga sekarang di GKI Sulung. Terima kasih atas

bimbingan, nasihat, teladan, dan tumpangannya di rumah Sutorejo selama saya kuliah di ITS ☺.

13. Rekan-rekan Komisi Pemuda GKI Sulung: BPH Komisi Pemuda (Welly, Ce Olivia TG, Cing Cing, Mbak Yanti, Citra, Mung Mung, Heidy), Tim KI Persekutuan Pemuda (Mbak Honny, Mbak Sofi, Lukas, Yuliana, Dewi, Ronny, Eko, Simon), rekan-rekan pemuda lainnya yang tidak dapat disebutkan satu per satu. Tidak lupa juga terima kasih kepada para Hamba Tuhan GKI Sulung: Pdt. Nathanael C. dan keluarga, Pdt. B.J. Siswanto dan keluarga, Pdt Ruth R. N., dan Pnt Soewandi T dan keluarga. Serta terima kasih kepada para staf kantor GKI Sulung: Bapak Soeparno, Bapak Agus T. Wullur, Bapak Harry P., Mbak Erna, Mbak Juliet, serta Ibu Hanifah *(Cik Ai)*, Bapak Wardoyo, Bapak Prapto, dll.
14. Para teman-temanku alumni SMU Kr. Petra 3: Danady (terima kasih atas VGA Card-nya), Tinna L. (terima kasih atas flash disknya), JePe, Coro, Lily, Gita, Carlini, Lidya, Sendi, Soe, Silvie. Terima kasih kalian semua telah menjadi sahabat-ku. Dan penulis berharap kita tetap jadi sahabat sampai kapan pun. Ok…teman-teman kapan kita jalan-jalan ?
15. Hathfi Bahrial Hakim *(a.k.a. Pipia / Piyo)*, sebagai rekan penulis dalam mengerjakan Tugas Akhir ini. Terima kasih atas persahabatan, semangat, motivasi, advice, ilmu, dan terutama ajakanmu dalam mengerjakan Tugas Akhir ini. *Thanks for everything, Bro! I'll never forget you!*
16. Sahabat-sahabatku di TC: Robby *(terima kasih atas kebersamaan, khususnya waktu sebulan lebih KP di Cilacap)*, Yogi/Sempal, Dhewy

"Cute", Widya "Cakep", Lufi "Imut", Rosdiana/Ossy, Melinda, Rinda, Arief/Karpo, Nina & Phe, Connie, and *"the three stooges in AJK"*: Willy Jansen, Pras, Rudy. *I'll never forget you all, guys!*

17. Teman-teman angkatan 2001 (C11) sesama TA-ers : Eva, Deddy, Azizah, Siti, Yeni, Sigit "Bulu", Sigit "Kentis", Eko, Ali, Firman "Selong", Dwi, Galih "kecil", Ria, Haris, Mas Choy, dan teman-teman C11 lainnya : Fanni, Yayan, Cukris, Tendra, Ricky, Omadhi, dll.
18. Laboratorium Komputing beserta semua komputer (Asteroid, Andromeda/Nebula, Orion, Bimasakti, Antariksa, Cakrawala, dll) dan fasilitas lain di dalamnya, dimana penulis boleh mengerjakan tugas kuliah dan Tugas Akhir di laboratorium ini selama hampir 2 tahun.
19. Komunitas Komputing yang telah lulus maupun yang belum: Mas Gershom, Mas Kilay, Mas Gus Pras, Mas Purna, Mbak Faida, Mbak Luluk, Mbak Dwi, Mbak Nuning, Mbak Ningrum, Mbak AAK, Mbak Adeta, Mas Rai, Mas Cherry, Mas Indie, Mas AW, Mas Alwi, Mas Anton, Mbak Anis R., Mas Rahde, Ko Toni, Edwin *(koppler)*, Diana, Alberth, Frans, Asnita, Bambang, Ita.
20. Para Senior di TC yang telah menjadi teladan bagi penulis : Ko Gae Siang, Ko Setiady, Ko Yos Nugroho, Ko Ronald, Ko Budianto, Ko Leo, Ko Chendra, Ko Arie Hintono, Ce Melissa, Ce Monita, Mas Ferdian, dan Mas Okhi "nguik-nguik".
21. Teman-teman di PJTC atau PD FTIf : Mas Nunung, Bang Waren, Mas Bimo & Mbak Rachel, Mas Alby, Ko Pieter, Mbak Intan, Mas Edward,

Mas Victor, Ko David, Mbak Keke, Mbak Maria, Leris, Dwight, Bang Oon, Rolly, Roy Nugroho (thanks for the Master Theorem), Egawati, Seven, Ira, Fernando "Ucok" Sebayang, Lian, Kario, James, Ruth, Laurence, *and many more...*

22. Teman-teman di PMK: Willy Singgih Chandrasasmita (thanks atas doa dan kehadiranmu wakut malam-malam sebelum sidang TA), Lidya Tekkim, Esther, Pevi, Richard TI, Theo, Alfred, dll.
23. Temen-temen seperguruan : Mas Decki, Mbak Suci, Mbak Nindi, Mbak Titin, Mbak Wida, Mbak Alifah, Mas Ashar. Semangat-semangat!!!
24. Teman-teman angkatan 2004 yang menemani penulis kuliah PIL: Intan, Dea, Naj'wa, Chetul, Anggi, Lisa, Ayik Tembem, Christian, Baskoro, dll.
25. Teman-teman yang sering kali menemani lewat chatting Yahoo Messenger dan MSN Messenger: Nathania Limanto, Wiyani, Ce Ester, Louise, Imelda, Ulma Maul, Vicky, Merli my cousin, Leeysa, Paulina, Ko Irwantoro, Ko Welly Tjan, dll.
26. Bu Fee Bing dan Ibu Lanny H., sebagai konselor yang telah meluangkan waktunya untuk mendengarkan permasalahan dari penulis.
27. Para hamba Tuhan yang sering memberi nasihat kepada penulis: Ibu Pdt. Anni P. Saleh, Kak Tridyah, Kak Tata, dan Ko Jeffry Sudirgo.
28. Ko Yudi, selaku pengajar komputer pertama kali bagi penulis. Terima kasih telah membuatku menyenangi komputer, Kapan nih main Fifa-nya diteruskan?
29. Semua yang kelupaan disebut. Jangan marah dan sakit hati ya !

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

Pada bab ini akan dibahas mengenai latar belakang dan tujuan dari pembuatan tugas akhir, menentukan permasalahan yang akan dipecahkan beserta batasan dari penyelesaiannya. Kemudian metodologi yang digunakan dalam penyelesaian permasalahan serta sistematika dalam penulisannya.

1.1. Latar Belakang

Perhitungan jarak terdekat dan pembuatan lintasan pada suatu permukaan merupakan permasalahan umum dalam dunia ilmu pengetahuan dan teknik khususnya *Computational Geometry*. Masalah ini muncul pada beberapa aplikasi seperti perencanaan gerakan robot, navigasi *terrain*, dan Sistem Informasi Geografik.

Geodesic Distance adalah jarak terdekat antara pasangan titik pada permukaan objek 3 dimensi. Jarak ini dihitung tanpa melewati bagian dalam dari objek 3 dimensi tersebut. *Geodesic Path* adalah lintasan pada permukaan objek 3 dimensi yang jaraknya direpresentasikan oleh *Geodesic Distance*.

Ada beberapa algoritme yang telah dikembangkan untuk perhitungan *Geodesic Distance* ini. Algoritme-algoritme tersebut mempunyai tingkat kompleksitas yang berbeda-beda. Salah satu algoritme yang paling awal dikembangkan untuk menghitung *Geodesic Distance* ini adalah *Dijkstra's-like algorithm* yang dikembangkan oleh Mitchell pada tahun 1987. Algoritme

Dijkstra's-like ini berjalan pada tingkat kompleksitas waktu $O(m^2 \lg m)$ dan tingkat kompleksitas ruang $O(m^2)$ dimana m adalah jumlah dari *edge* pada *mesh*. Pada tahun 1999, Kapoor juga mengembangkan algoritme yang mirip dengan *Dijkstra's-like Algorithm* yang dikembangkan oleh Mitchell, tetapi dengan struktur data yang lebih efisien sehingga algoritme tersebut dapat berjalan pada tingkat kompleksitas waktu $O(n \lg^2 n)$ dimana n adalah jumlah titik (*vertices*) pada *mesh*. Pada tahun 1990, Chen dan Han mengembangkan algoritme yang membangun struktur data berdasar pada pembentangan permukaan (*surface unfoldings)*. Algoritme tersebut berjalan pada tingkat kompleksitas waktu $O(n^2)$ dan tingkat kompleksitas ruang $O(n)$. Pada tahun 1998, Sethian dan Kimmel menggunakan *Fast Marching Method* (FMM) untuk menentukan fungsi jarak dari satu titik ke semua titik lainnya pada permukaan dengan tingkat kompleksitas waktu $O(n \lg n)$. Sampai sekarang mungkin algoritme *Fast Marching Method* inilah yang mempunyai tingkat kompleksitas waktu yang paling cepat. Tetapi tidak seperti algoritme-algoritme lainnya, algoritme *Fast Marching Method* ini tidak menghasilkan *Geodesic Distance* dan *Geodesic Path* yang sebenarnya, melainkan hanya perkiraannya saja.

Oleh karena itu dalam tugas akhir ini akan dilakukan penerapan suatu algoritme perhitungan *geodesic distance* pada permukaan objek 3 dimensi yaitu algoritme *Fast Marching Method* untuk dianalisis tingkat keakuratan dan tingkat efisiensinya.

1.2. Tujuan Pembuatan Tugas Akhir

Tujuan dari pembuatan tugas akhir ini adalah merancang dan membangun perangkat lunak yang menerapkan dan mengimplementasikan algoritme *Fast Marching Method* untuk perhitungan *Geodesic Distance* dan pembuatan *Geodesic Path* pada permukaan objek 3 dimensi. Setelah itu menganalisis kinerja algoritme tersebut.

1.3. Permasalahan

Berdasarkan latar belakang yang telah diuraikan sebelumnya, permasalahan yang akan diselesaikan dalam tugas akhir ini adalah sebagai berikut:

1. Bagaimana mengimplementasikan algoritme *Fast Marching Method* sebagai algoritme untuk menghitung *Geodesic Distance*.
2. Bagaimana membuat *Geodesic Path* dari *Geodesic Distance* yang telah dihasilkan oleh algoritme *Fast Marching Method.*
3. Bagaimana cara menggunakan file dengan format .PLY sebagai data *triangular mesh* dalam aplikasi visualisasi objek tiga dimensi.
4. Bagaimana membuat struktur data yang tepat untuk implementasi algoritme *Fast Marching Method.*
5. Bagaimana cara menerapkan fungsi fungsi library yang disediakan OpenGL dalam aplikasi Visualisasi Objek 3 Dimensi.
6. Bagaimana mengadakan analisis dan uji coba terhadap implementasi algoritme *Fast Marching Method.*

1.4. Batasan Permasalahan

Dari permasalahan-permasalahan di atas, maka batasan dalam tugas akhir ini adalah:

1. Data Objek yang digunakan adalah data dengan format file `.ply` yang merupakan data *polygon mesh* dan dikhususkan untuk data poligon segitiga (*triangular mesh*).
2. Algoritme yang dianalisis adalah *Fast Marching Method.*
3. Graphic Library yang digunakan adalah Open Graphic Library (OpenGL).
4. Bahasa Pemrograman yang digunakan adalah menggunakan C++ dengan Compiler dan Tools Microsoft Visual C++ 6.0.
5. Batasan dalam implemetasi algoritme adalah komputasi sekuensial.
6. Batasan parameter yang digunakan untuk melakukan percobaan adalah besar data dan lama waktu pemrosesan

1.5. Metodologi Tugas Akhir

Metodologi yang digunakan dalam penyusunan tugas akhir ini diantaranya:

1. Studi literatur dan pemahaman algoritme

   Sebagai langkah awal, pada tahap ini dilakukan pengumpulan literatur dan mempelajari konsep dan teori algoritme *Fast Marching Method.*.
2. Perumusan masalah dan perumusan penyelesaiannya

   Langkah berikutnya adalah pendefinisian masalah, batasan-batasan masalah, serta langkah-langkah penyelesaian yang mengarah pada

perangkat lunak yang akan dikembangkan.

3. Perancangan perangkat lunak

   Langkah yang dilakukan pada tahap ini adalah melakukan perancangan data, perancangan antar muka, perancangan proses perhitungan *Geodesic Distance* dan proses pembuatan *Geodesic Path*..

4. Pembuatan perangkat lunak

   Pada tahap ini dilakukan proses implementasi dari perancangan perangkat lunak yang telah dibuat pada tahap sebelumnya.

5. Pengujian dan revisi program

   Pada tahap ini dilakukan evaluasi dan perbaikan dari program yang telah dibuat.

6. Penulisan naskah Tugas Akhir

   Pada tahap akhir dari serangkaian metodologi ini, dilakukan penulisan dokumentasi dari tahapan konsep, perancangan, sampai tahap akhir, yaitu penerapannya.

1.6. SISTEMATIKA PENULISAN

Metodologi yang digunakan dalam Penyusunan Tugas akhir ini adalah sebagai berikut :

1. **BAB I Pendahuluan**

   Pada bagian ini dijelaskan tentang latar belakang yang mendasari tugas akhir ini, tujuan serta manfaat tugas akhir, batasan masalah, metodologi pembuatan tugas akhir, serta sistematika penulisan buku tugas akhir ini.

2. **BAB II Dasar Teori**

   Bab ini membahas mengenai dasar teori yang digunakan pada tugas akhir ini diantaranya mengenai algoritme *Fast Marching Method* yang digunakan sebagai algoritme perhitungan *Geodesic Distance*, proses pembuatan *Geodesic Path*, struktur data Heap sebagai struktur data yang digunakan dalam algoritme *Fast Marching Method*, dan format file PLY sebagai format file masukan dari perangkat lunak.

3. **BAB III Perancangan Perangkat Lunak**

   Bab ini membahas mengenai perancangan perangkat lunak perhitungan *Geodesic Distance* dan pembuatan *Geodesic Path* pada permukaan objek 3 dimensi meliputi perancangan data, perancangan proses dan perancangan antar muka.

4. **BAB IV Implementasi Perangkat Lunak.**

   Bab ini membahas mengenai implementasi dari perancangan yang telah dibuat pada bab III meliputi implementasi struktur data, implementasi proses perhitungan *Geodesic Distance*, implementasi proses pembuatan *Geodesic Path*, dan implementasi antar muka.

5. **BAB V Uji Coba dan Evaluasi**

   Bab ini membahas mengenai uji coba dan evaluasi yang dilakukan terhadap perangkat lunak yang telah dibangun.

6. **BAB VI Penutup**

   Bab ini menjelaskan mengenai kesimpulan dan saran dari keseluruhan Tugas Akhir ini.

BAB II
DASAR TEORI

Pada bab ini akan dibahas mengenai teori-teori penunjang dalam perancangan dan pembuatan aplikasi perhitungan *Geodesic Dsitance* dan pembangunan *Geodesic Path* pada permukaan objek tiga dimensi. Pembahasan akan dimulai dari pengenalan mengenai *geodesic distance* dan *geodesic path,* kemudian dilanjutkan *Fast Marching Method*, algoritme yang digunakan untuk melakukan perhitungan *Geodesic Distance* pada permukaan objek tiga dimensi dan Pembuatan *Geodesic Path.* Pada bagian selanjutnya akan dibahas mengenai format file PLY yang digunakan sebagai data masukan dan keluaran perangkat lunak.

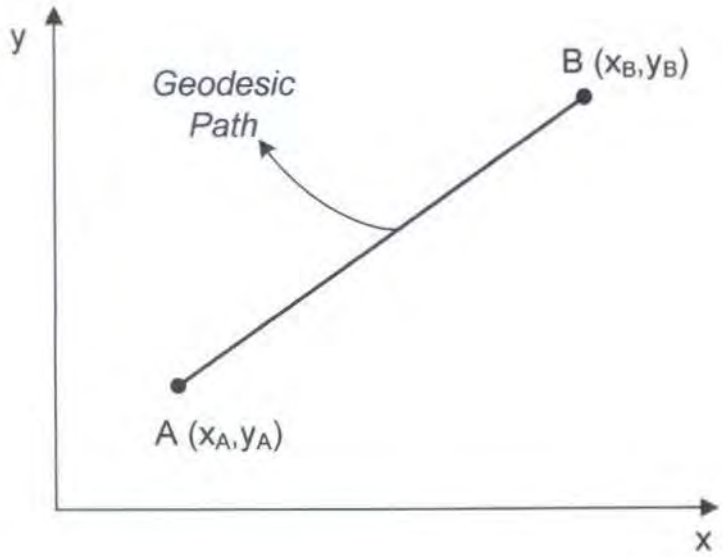
2.1. GEODESIC DISTANCE DAN GEODESIC PATH

Geodesic Distance adalah jarak terdekat antara pasangan titik pada permukaan objek 3 dimensi. Jarak ini dihitung tanpa melewati bagian dalam dari objek 3 dimensi tersebut. *Geodesic Path* adalah lintasan pada permukaan objek 3 dimensi yang merepresentasikan *Geodesic Distance*.

Geodesic Distance antara 2 titik pada suatu permukaan bidang datar adalah jarak Euclidian. Jarak Euclidian dinyatakan oleh rumus sebagai berikut :

$$d = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2} \qquad 2.1$$

dimana d adalah jarak Euclidian antara titik A (x_A, y_A) dan titik B (x_B, y_B).



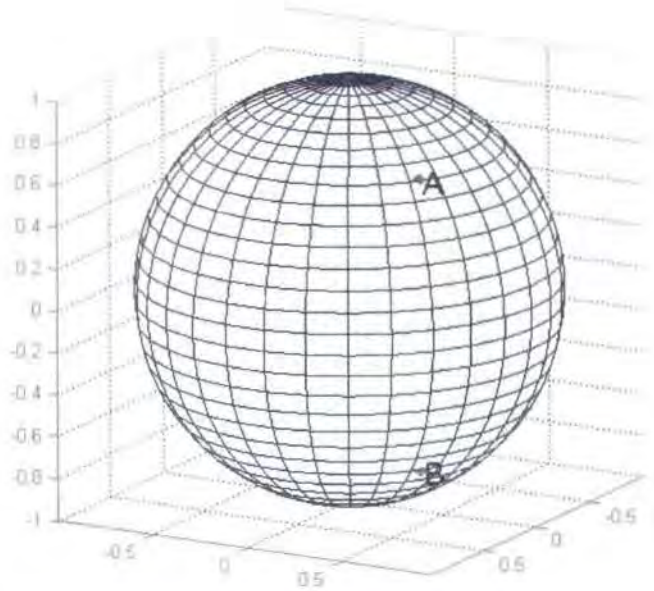
Gambar 2-1 Geodesic Path Pada Bidang Datar (bidang XY)

Geodesic Path yang merepresentasikan *Geodesic Distance* dari 2 titik pada permukaan bidang datar adalah sebuah garis lurus yang menghubungkan titik A dan titik B (gambar 2-1).

Geodesic Distance antara 2 titik pada suatu permukaan bola (*sphere*), dapat dihitung menggunakan rumus sebagai berikut :

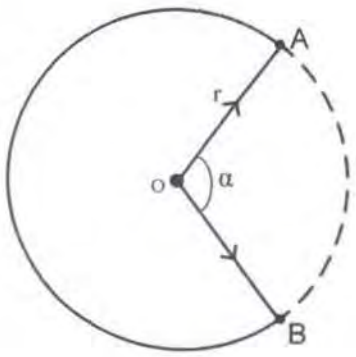
$$d = r.\alpha \qquad 2.2$$

dimana r adalah panjang jari-jari bola tersebut dan α adalah besar sudut pusat yang dibentuk oleh 2 vektor dari 2 titik pada permukaan bola tersebut.



Gambar 2-2 Permukaan Bola (Sphere) Dilihat Secara 3 Dimensi

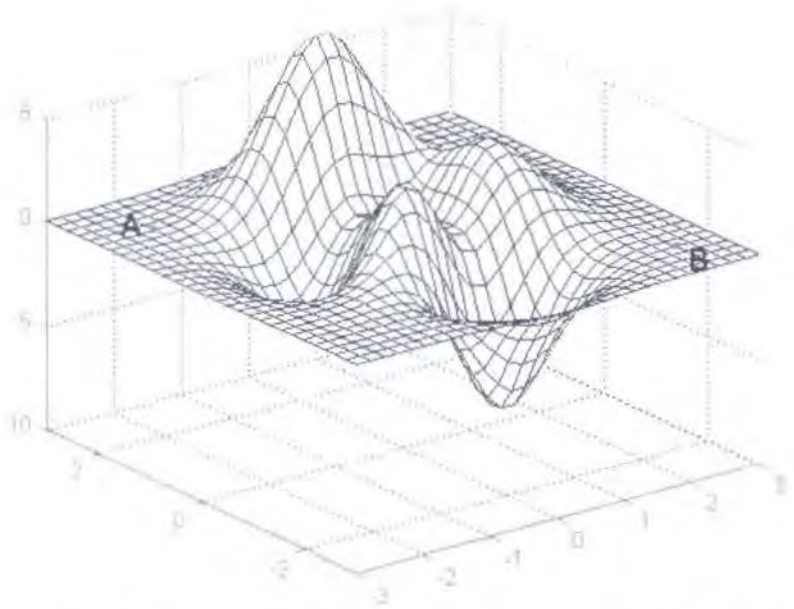
Geodesic Path yang merepresentasikan *Geodesic Distance* dari 2 titik pada suatu permukaan bola (*sphere*) adalah berupa busur dari lingkaran yang melalui 2 titik pada permukaan tersebut dan berpusat pada pusat bola.



Gambar 2-3 Penampang Bola (Sphere) Dilihat Secara 2 Dimensi

Pada Gambar 2-3 di atas tampak bola secara 2 dimensi (dilihat dari depan). *geodesic path* yang merepresentasikan *geodesic distance* antara titik A dan B adalah busur AB yang ditunjukkan dengan garis putus-putus.

Geodesic Distance antara 2 titik pada suatu permukaan objek 3 dimensi terutama objek 3 dimensi yang tidak beraturan sulit sekali untuk dihitung. Hal ini disebabkan karena kontur dari permukaan objek 3 dimensi bisa bermacam-macam sekali bentuknya.



Gambar 2-4 Contoh Permukaan Objek 3 Dimensi yang Tidak Beraturan

Seperti contoh permukaan objek 3 dimensi pada Gambar 2-4, tentulah sulit untuk menghitung *Geodesic Distance* antara titik A dan B dengan tepat, karena kontur dari permukaan objek tersebut tidaklah rata dan teratur. Untuk memecahkan masalah tersebut dilakukan beberapa pendekatan secara numerik, salah satunya adalah Algoritme *Fast Marching Method* yang akan dijelaskan pada sub bab berikut.

2.2. FAST MARCHING METHOD

Algoritme perhitungan *Geodesic Distance* pada permukaan objek 3 dimensi merupakan topik umum dalam dunia *Computational Geometry* atau Ilmu ukur ruang. Banyak penelitian masih dilakukan untuk menemukan algoritme perhitungan *Geodesic Distance* yang akurat dan efektif.

Algoritme yang paling akhir dikembangkan adalah *Fast Marching Method*. *Fast Marching Method* ini diperkenalkan oleh James A. Sethian, seorang Profesor Matematika dari University of California, Barkeley, USA. Algoritme ini dapat digunakan secara luas dalam berbagai bidang ilmu, antara lain untuk menyelesaikan permasalahan : *shape offseting, shape from shading, shape detection,* pengembangan *photolithographic,* perhitungan *first arrivals in seismic travel times, optimal path planning around obstacles,* perhitungan jarak dari kurva yang komplek dan permukaan, pembuatan lintasan terpendek pada permukaan, dan lain-lain. Secara khusus algoritme *Fast Marching Method* untuk masalah perhitungan *Geodesic Distance* dan pembuatan *Geodesic Path* pada permukaan *triangular mesh* dikembangkan oleh James A. Sethian dan Ron Kimmel, algoritme yang dikembangkan dikenal dengan istilah *Fast Marching Method on Triangulated Domain (FMM on TD).*

Dalam permasalahan perhitungan *Geodesic Distance* dan pembuatan *Geodesic Path*, secara matematis tujuan dari algoritme *Fast Marching Method* ini adalah mencari pendekatan penyelesaian persamaan Eikonal :

$$|\nabla T|F = 1 \qquad 2.3$$

dengan T adalah fungsi jarak dari suatu titik terhadap titik awal *(starting vertex)*, dimana nilai T pada titik awal sama dengan 0 dan F adalah fungsi kecepatan dari propagasi, dimana F dapat berupa *cost* (biaya) yang dibutuhkan untuk berpindah dari satu titik ke titik lainnya, tetapi dalam permasalahan geometri yang hanya mempertimbangkan jarak saja nilai $F = 1$.

Inti dari algoritme *Fast Marching Method* ini adalah melakukan propagasi menyebar / maju *(front propagation)* dari sebuah titik sebagai *starting vertex* ke segala arah yang mungkin. Setiap bergerak maju algoritme ini selalu menghitung dan menyimpan nilai jarak suatu titik terhadap titik awal *(starting vertex)*, nilai jarak tersebut disimpan sebagai properti dari titik tersebut. Tiap titik diproses, semakin jauh titik tersebut nilai jaraknya menjadi semakin besar. Proses propagasi maju dari algoritme *Fast Marching Method* ini dapat dianalogikan sebagai kobaran api yang membakar padang rumput, kobaran api ini berjalan maju dengan kecepatan konstan ke segala arah membakar rumput yang belum terbakar. Proses "pembakaran" ini dimulai pada titik awal dan bergerak maju "membakar" titik-titik lain di sebelahnya. Titik yang telah "terbakar" berarti titik tersebut telah diproses atau telah dihitung nilai jaraknya dari titik awal. Proses ini berhenti sampai titik akhir *(end vertex)* telah "terbakar" atau telah dihitung nilai jaraknya.

Algoritme *Fast Marching Method on Triangulated Domain* ini mirip dengan algoritme Dijkstra yang digunakan untuk memecahkan masalah *single-source shortest paths* pada sebuah graf berbobot dan tak berarah *(weighted and*

nondirected graphs). Idenya adalah mendeskripsikan permukaan *triangular mesh* sebagai sebuah graf, dimana sisi dari segitiga adalah sebagai garis penghubung berbobot dan tak berarah *(nondirected weighted edge)* yang menghubungkan 2 *vertex*. *Vertex* pada graf dinyatakan oleh titik pada ruang 3 dimensi. Nilai bobot *(weight)* dari suatu *edge* sama dengan jarak Euclidian antara 2 titik dihubungkan oleh *edge* tersebut, $w(e_{v_i,v_j}) = d(v_i, v_j)$.

Algoritme Dijkstra terdiri dari langkah-langkah berikut :

Inisialisasi : Ambil v_0 menjadi *source vertex*, nilai $T(v_0) = 0$ dan tetapkan pada semua *vertex* lainnya $T(v) = \infty$.

Langkah 1 : *Update* semua *vertex* v_i yang terhubung dengan v_0 oleh sebuah *edge* melalui cara : $T(v_i) = \min\left(T(v_i), T(v_i) + w(e_{v_i,v_j})\right)$.

Langkah 2 : Masukkan *vertex* yang baru di-*update* tersebut ke dalam struktur data *min-heap*. Jika *vertex* tersebut telah ada di dalam *min-heap*, maka hanya *update* lokasi *vertex* tersebut di dalam *min-heap*.

Langkah 3 : Jika *min-heap* telah kosong, keluar dari proses, jika tidak pindahkan *vertex* pada puncak dari struktur *min-heap* dan sebut *vertex* tersebut sebagai v_0.

Langkah 4 : Kembali ke langkah 1.

Perbedaan antara algoritme *Fast Marching Method on Triangulated Domain* dengan algoritme Dijkstra adalah pada langkah *update* (langkah 2). Pada algoritme *Fast Marching Method on Triangulated Domain*, untuk meng-*update* nilai dari suatu titik digunakan *Update Sethian's Method*. Metode ini bertujuan

untuk menghitung nilai *T* dari suatu titik berdasarkan nilai *T* dari 2 titik lainnya yang berada pada segitiga yang sama.

Penjelasan langkah-langkah algoritme *Fast Marching Method on Triangulated Domain* akan dijelaskan secara detail pada sub bab berikutnya.

2.2.1 Karakteristik Fast Marching Method on Triangulated Domain

Masukan *(input)* dari algoritme *Fast Marching Method on Triangulated Domain* adalah sebuah *triangular mesh* yaitu representasi permukaan *(surface)* 3 dimensi yang dibentuk oleh segitiga-segitiga. *Triangular mesh* terdiri dari himpunan titik pada ruang 3 dimensi yang membentuk *triangulated surface*. Selain itu perlu juga masukan 2 buah titik dari himpunan titik tersebut sebagai titik awal *(starting vertex)* dan titik akhir *(end vertex)*.

Keluaran *(output)* dari algoritme *Fast Marching Method on Triangulated Domain* adalah *Geodesic Distance* antara titik awal dan titik akhir. Dan apabila digunakan untuk menyelesaikan *Single-source Shortest Paths Problem* maka keluaran yang dihasilkan adalah sebuah array dengan ukuran *n* (jumlah titik pada *mesh*). Array ini berisi nilai *T* dari semua titik pada *mesh*. Selain itu apabila digunakan untuk menyelesaikan *All Pairs Shortest Path Problem* maka keluaran yang dihasilkan adalah sebuah array dengan ukuran *n* x *n*.

Perlu diketahui bahwa pada algoritme ini, setiap titik pada *mesh* mempunyai beberapa properti yaitu :

- T, yang menyatakan nilai *geodesic distance* titik tersebut dari titik awal. Nilai T ini mempunyai *monotonicity property* yaitu semakin akhir titik tersebut dihitung nilai T-nya maka makin besar nilainya dibandingkan nilai T dari titik yang dihitung lebih awal. Jadi lama-kelamaan nilai T dari titik-titik yang diproses makin besar nilainya. Selain itu nilai T dari suatu titik hanya dapat di-*update* dengan nilai T baru yang lebih kecil dari nilai T sebelumnya, sehingga yang dihasilkan nantinya merupakan nilai T minimum, dengan kata lain merupakan jarak terdekat dari titik awal.
- Status, yang menunjukkan status dari titik tersebut.

  Ada 3 macam status, yaitu :

  - *Far*, artinya titik tersebut belum pernah diproses atau dihitung *geodesic distance*-nya.
  - *Close*, artinya titik tersebut pernah diproses, tetapi nilai *geodesic distance*-nya belum final. Nilai *geodesic distance* tersebut dapat berubah atau di-*update* lagi.
  - *Fix*, artinya titik tersebut telah diproses. Nilai *geodesic distance*-nya sudah Final sehingga nilai *geodesic distance* tersebut tidak perlu dan tidak dapat di-*update* lagi.

Dengan dibedakannya semua titik yang ada berdasarkan statusnya maka diperoleh 3 himpunan titik, yaitu : Himpunan *Far Vertex*, Himpunan *Closed Vertex*, dan Himpunan *Fixed Vertex*.

2.2.2 Tahap Inisialisasi

Sama halnya dengan algoritme Dijkstra, algoritme *Fast Marching Method on Triangulated Domain* mempunyai tahap inisialisasi. Berikut langkah-langkah dalam tahap inisialisasi :

- Semua titik yang ada statusnya ditetapkan menjadi *Far*.
- Nilai *geodesic distance* atau nilai T dari tiap titik ditetapkan menjadi tidak hingga *(infinite)*.
- Pada titik awal *(Starting vertex)*, nilai T ditetapkan = 0.
- Status titik awal ditetapkan menjadi *close*. Hal ini berarti memindahkan titik awal dari himpunan *far vertex* ke himpunan *closed vertex*.

2.2.3 Tahap Perulangan

Setelah tahap inisialisasi selesai dilakukan, maka algoritme *Fast Marching Method on Triangulated Domain* dilanjutkan dengan tahap perulangan. Tahap perulangan ini terus dilakukan sampai nilai *geodesic distance* dari titik akhir *(end vertex)* telah diperoleh atau dengan kata lain sampai status dari titik akhir menjadi *fix*.

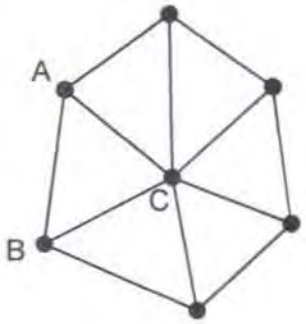
Berikut langkah-langkah dari tahap perulangan :

- Ambil titik *trial* yang merupakan titik pada himpunan *closed vertex* dengan nilai T terkecil / minimum.
- Tetapkan status titik *trial* tersebut menjadi *fix*.
- Tetapkan juga status tiap titik tetangga dari titik *trial* (titik yang terhubung oleh satu *edge* / garis penghubung dengan titik *trial*) yang bukan anggota

dari himpunan *fixed vertex* menjadi *close*. Hitung nilai T dari tiap titik tersebut. Apabila T yang baru dihitung lebih kecil dari T yang lama maka *update* nilai T dengan nilai T yang baru, jika tidak maka nilai T yang lama tidak perlu diganti. Langkah-langkah dalam menghitung nilai T akan dijelaskan pada sub bab berikutnya.

- Kembali ke langkah pertama tahap perulangan.

2.2.3.1 Menghitung Nilai T



Gambar 2-5 Titik C yang berada dalam satu segitiga dengan titik A dan B

Dalam tahap perulangan di sub bab sebelumnya, disebutkan pada langkah ketiga untuk menghitung nilai T dari tiap titik tetangga dari titik *trial*. Pada gambar 2-5, untuk menghitung nilai T dari suatu titik (sebut saja titik C) diperlukan nilai T dari 2 titik lainnya (sebut saja titik A dan B) yang berada dalam segitiga yang sama dengan titik yang ingin dihitung nilai T-nya (titik C). Nilai T dari titik-titik lain tersebut harus memenuhi $T(B) > T(A)$, jika tidak maka titik yang menjadi A dan B ditukar. Apabila titik tersebut mempunyai banyak segitiga, maka titik tersebut akan dihitung nilai T-nya sebanyak segitiga yang dimilikinya. Lalu diambil nilai T yang paling kecil untuk menjadi nilai *geodesic distance* titik tersebut dari titik awal.

Ada 3 kemungkinan yang dapat terjadi waktu menghitung nilai T pada suatu titik atau dengan kata lain waktu mencari T(C) dari T(A) dan T(B) yang diketahui nilainya, yaitu :

- Jika T(A) = ∞ dan T(B) = ∞, maka nilai T(C) tidak bisa dihitung dari T(A) dan T(B) tersebut.
- Jika T(A) < ∞ dan T(B) = ∞, maka nilai T di titik C dapat dihitung dengan menggunakan rumus :

$$T(C) = T(A) + b \qquad 2.4$$

dimana *b* adalah jarak euclidian antara titik A dan titik C.

- Jika T(A) < ∞ dan T(B) < ∞, maka nilai T di titik C dapat dihitung dari nilai T(A) dan T(B) dengan menggunakan proses *Update Sethian's Method* yang akan dijelaskan pada sub bab 2.2.3.2. Tetapi perlu diperhatikan terlebih dahulu bahwa sebelum melakukan proses *Update Sethian's Method*, sudut C pada segitiga ABC tersebut harus merupakan sudut lancip (sudut C $\leq 90^0$). Jika sudut C pada segitiga ABC bukan merupakan sudut lancip (sudut C > 90^0), maka harus dilakukan proses *Unfolding Triangles* terlebih dahulu untuk membagi sudut C itu menjadi 2 buah sudut lancip. Proses *Unfolding Triangles* akan dijelaskan pada sub bab 2.2.3.3.

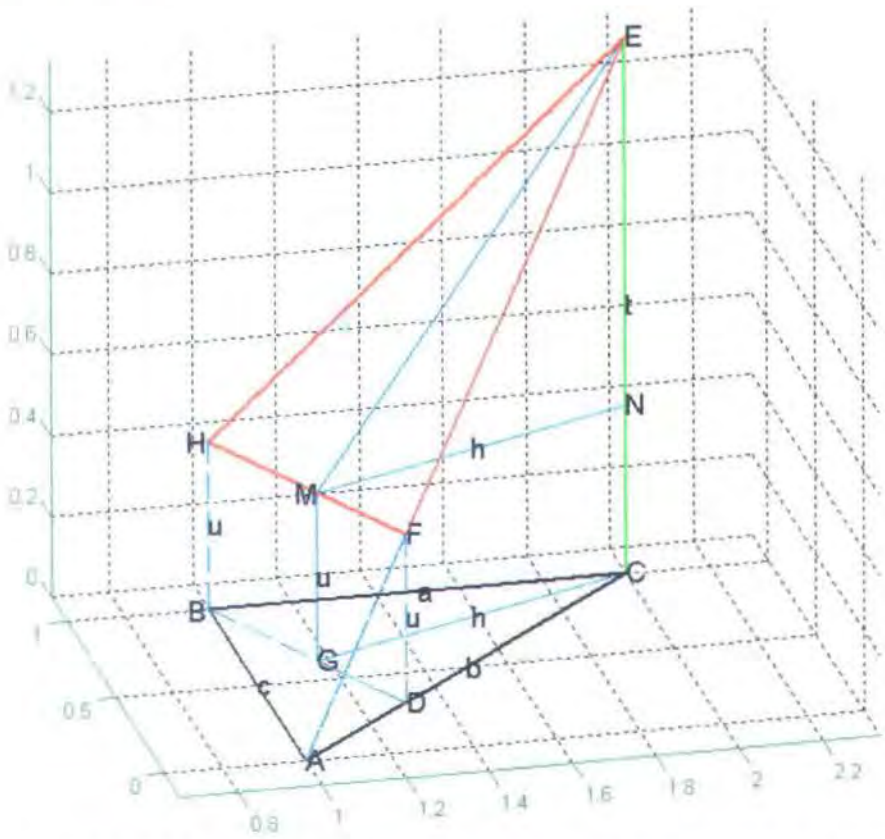
2.2.3.2 Proses Update Sethian's Method

Proses ini dilakukan waktu menghitung nilai T di suatu titik dengan bantuan nilai T dari 2 titik lainnya yang berada pada segitiga yang sama. Misalkan titik yang ingin dihitung nilai T nya adalah titik C dan 2 titik lain yang berada dalam

satu segitiga dengan titik C adalah titik A dan B, maka nilai T pada titik C dapat dihitung menggunakan rumus :

$$T(C) = T(A) + t \qquad 2.5$$

Tujuan dari *Update Sethian's Method* adalah untuk mencari nilai *t*. Caranya ialah dengan membangun sebuah bidang ΔEFH di atas ΔABC dimana ΔEFH tersebut mempunyai kemiringan terhadap ΔABC = 1. Sudut EMN = 45^0. Didefinisikan juga nilai $u = T(B) - T(A)$, panjang sisi BC = *a*, panjang sisi AC = *b*, panjang sisi AB = *c*.



Gambar 2-6 Konstruksi ΔEFH yang mempunyai gradient terhadap ΔABC = 1

Dari gambar 2-6 di atas, maka dapat dirumuskan bahwa :

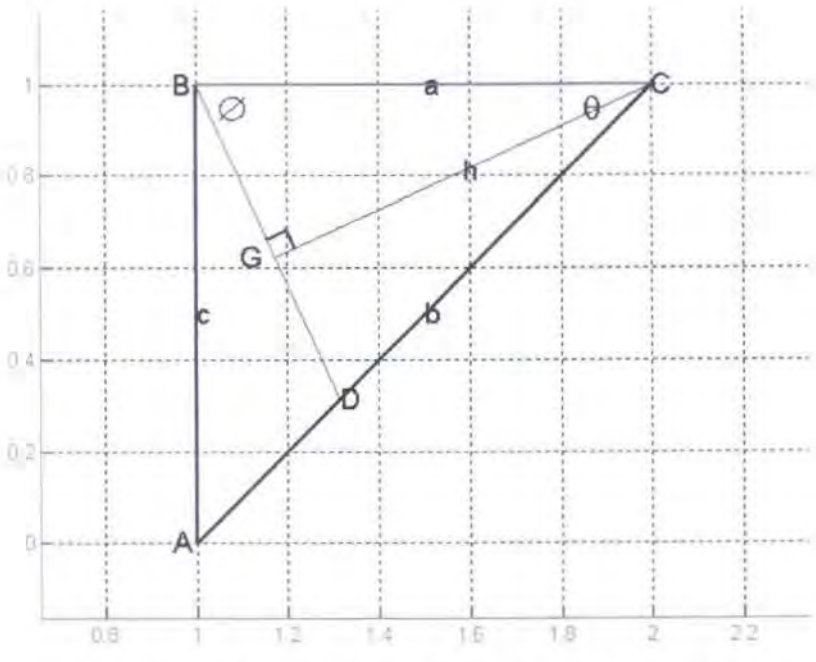
$$\frac{t-u}{h} = w \qquad 2.6$$

dimana h adalah panjang garis MN dan w adalah kemiringan ΔEFH terhadap ΔABC, jadi $w = \tan \angle EMN = 1$.

Kemudian panjang sisi AD dapat ditulis sebagai :

$$\frac{t}{b} = \frac{DF}{AD} = \frac{u}{AD}$$

$$AD = \frac{bu}{t}$$



Gambar 2-7 ΔABC dilihat dari atas (secara 2 dimensi)

Dari gambar 2-7 di atas, maka panjang sisi CD dapat dirumuskan dengan menggunakan panjang sisi AD, yaitu :

$$CD = b - AD = b - \frac{bu}{t} = \frac{b(t-u)}{t} \qquad 2.7$$

Setelah itu nilai h (panjang garis GC yang tegak lurus dengan garis BD) dapat dihitung melalui proses dalil Cosinus dan dalil Sinus pada segitiga BCD terlebih dahulu. Didefinisikan $\phi = \angle CBD$ dan $\theta = \angle BCD$.

Dengan Dalil Cosinus pada ΔBCD diperoleh :

$$BD^2 = a^2 + CD^2 - 2aCD\cos\theta$$

Kemudian dengan Dalil Sinus pada ΔBCD diperoleh :

$$\frac{\sin\phi}{\sin\theta} = \frac{CD}{BD}$$

$$\sin\phi = \frac{CD}{BD}\sin\theta$$

Akhirnya dengan menggunakan ΔCBG, maka diperoleh :

$$h = a\sin\phi$$

$$h = a\frac{CD}{BD}\sin\theta$$

$$h = \frac{a.CD\sin\theta}{\sqrt{a^2 + CD^2 - 2aCD\cos\theta}} \qquad 2.8$$

Lalu dengan menggunakan persamaan 2.6 dan memasukkan persamaan 2.7 ke persamaan 2.8 diperoleh :

$$h = \frac{t-u}{w}$$

$$\frac{t-u}{w}=\frac{a\left(\frac{b(t-u)}{t}\right)\sin\theta}{\sqrt{a^2+\left(\frac{b(t-u)}{t}\right)^2-2a\left(\frac{b(t-u)}{t}\right)\cos\theta}} \qquad 2.9$$

Akhirnya dari persamaan 2.9 akan diperoleh persamaan kuadrat untuk nilai t :

$$\left(a^2+b^2-2ab\cos\theta\right)t^2+2bu(a\cos\theta-b)t+b^2(u^2-w^2a^2\sin^2\theta)=0$$

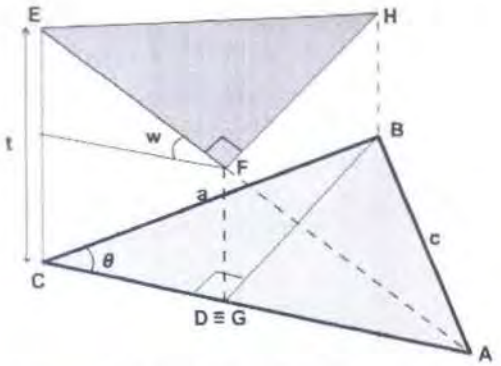
$$c^2t^2+2bu(a\cos\theta-b)t+b^2(u^2-w^2a^2\sin^2\theta)=0 \qquad 2.10$$

Persamaan kuadrat untuk nilai t di atas dapat diselesaikan dengan menggunakan rumus *abc*, sebagai berikut :

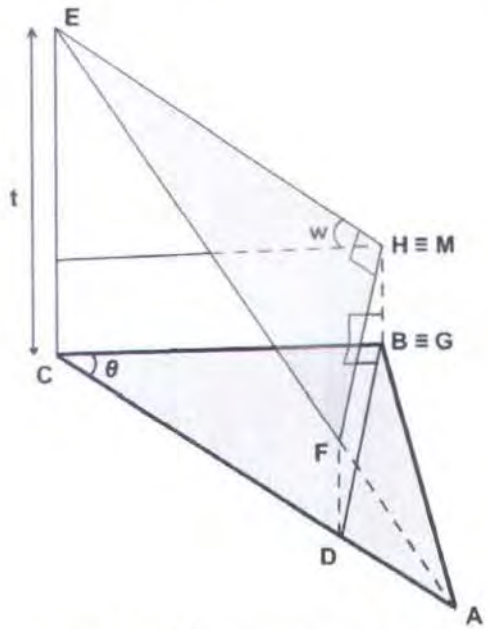
Jika diketahui persamaan kuadrat nilai x: $ax^2+bx+c=0$

Maka penyelesaiannya : $x_1, x_2=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$

Untuk menjaga konsistensi dan *monotonicity property*, nilai t harus selalu bernilai positif. Selain itu titik update G harus berada di dalam segitiga ABC. Titik G yang dihasilkan harus selalu ada pada garis BD. Ketika G ≡ D, maka $\angle BDC=\frac{\pi}{2}$ dan panjang $CD=a\cos\theta$, seperti yang digambarkan pada gambar 2-8. Pada batasan lain, jika G ≡ B, maka $\angle CBD=\frac{\pi}{2}$ dan panjang $CD=\frac{a}{\cos\theta}$, seperti yang digambarkan pada gambar 2-9.



Gambar 2-8 ΔABC jika titik update G ≡ D



Gambar 2-9 ΔABC jika titik update G ≡ B

Jadi agar titik G, tetap berada di dalam segitiga maka $CD = \frac{b(t-u)}{t}$ (dari persamaan 2.7) harus memenuhi persamaan :

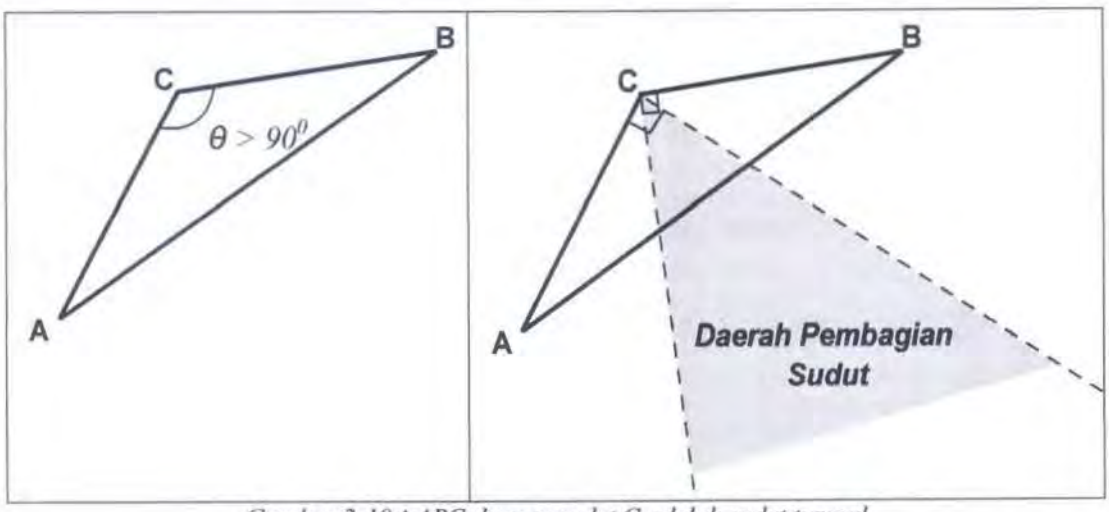
$$a\cos\theta \le \frac{b(t-u)}{t} \le \frac{a}{\cos\theta} \qquad 2.11$$

Selain itu solusi nilai t dari persamaan 2.10 harus memenuhi : $u < t$. Jika kedua syarat tersebut terpenuhi maka untuk menghitung nilai T pada titik C

digunakan $T(C) = \min\{T(C), T(A) + t\}$, jika tidak terpenuhi maka nilai T pada titik C dihitung dengan $T(C) = \min\{T(C), T(A) + b, T(B) + a\}$

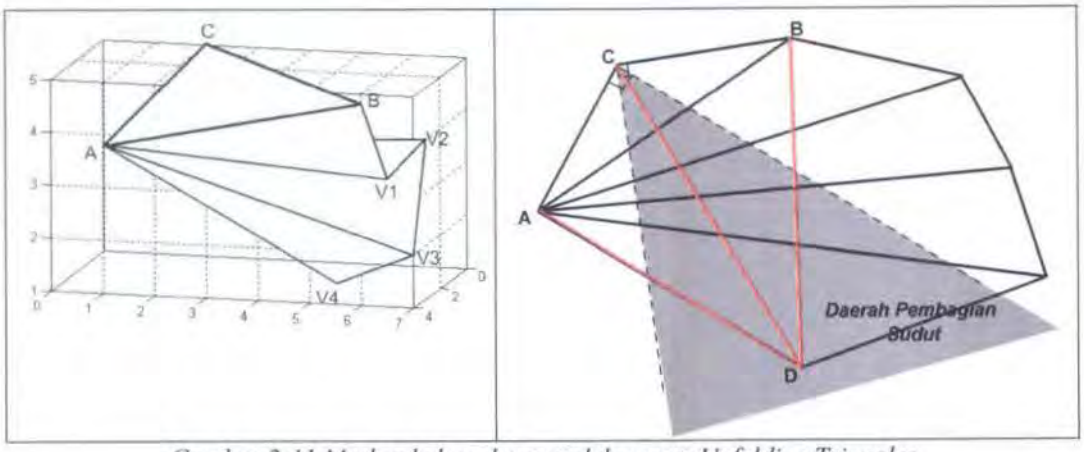
2.2.3.3 Proses Unfolding Triangles

Proses *Update Sethian's Method* yang telah dijelaskan pada sub bab sebelumnya ini hanya dapat dilakukan jika sudut C pada ΔABC besarnya tidak lebih dari 90^0 atau dengan kata lain sudut C adalah sudut lancip. Ini merupakan syarat mutlak karena jika sudut C lebih besar dari 90^0 atau sudut C merupakan sudut tumpul akan mengakibatkan nilai $\cos\theta$ negatif (gambar 2-10 kiri).



Gambar 2-10 ΔABC dengan sudut C adalah sudut tumpul

Untuk itu jika sudut C pada suatu segitiga merupakan sudut tumpul, maka perlu dilakukan proses *Unfolding Triangles* untuk membagi sudut C tersebut menjadi 2 buah sudut lancip.



Gambar 2-11 Mesh sebelum dan sesudah proses Unfolding Triangles

Ide dari proses *Unfolding Triangles* ini adalah membuat daerah pembagian sudut (gambar 2-10 kanan), lalu memperluas daerah segitiga mula-mula (ΔABC) dengan membentangkan (unfolding) segitiga-segitiga yang bersebelahan sampai diperoleh sebuah titik D yang berada pada daerah pembagian sudut (gambar 2-11 kanan). Dengan garis yang menghubungkan titik D dan titik C, maka sudut C tersebut telah dibagi menjadi 2 buah sudut lancip yaitu : $\angle ACD$ dan $\angle BCD$.

Setelah proses *Unfolding Triangles* ini selesai, maka dilakukan proses *Update Sethian's Method* pada ΔACD dan dilanjutkan pada ΔBCD untuk menghitung nilai T, nilai T yang terkecil diambil untuk menjadi nilai T(C).

2.2.4 Kompleksitas Algoritme Fast Marching Method

Algoritme *Fast Marching Method on Triangulated Domain* ini mempunyai kompleksitas waktu $O(n \lg n)$ dimana n adalah jumlah titik yang ada, nilai kompleksitas ini didapat dari :

- Waktu untuk meng-*update* nilai T dari semua titik yang ada = $O(n)$

Algoritme *Fast Marching Method* akan menghitung semua nilai T dari tiap titik yang ada sampai nilai T pada titik akhir (*end vertex*) telah diperoleh dan status titik akhir tersebut telah menjadi *fix*. Proses meng-*update* nilai T dari tiap titik tersebut berjalan dengan kompleksitas waktu $O(n)$.

- Waktu untuk memelihara struktur data min-priority queue dari himpunan *close vertex* = $O(\lg n)$

  Di dalam tiap tahap perulangan dilakukan operasi-operasi push, pop dan decrease-key terhadap min-priority queue yang menyimpan himpunan *close vertex*. Tiap operasi tersebut berjalan dengan kompleksitas waktu $O(\lg n)$. Penjelasan lebih detail mengenai struktur data min-priority queue ada pada sub-bab 2.4.

2.3. Pembuatan Geodesic Path

Geodesic Path adalah lintasan terpendek pada permukaan objek 3 dimensi yang menghubungkan 2 buah titik pada permukaan tersebut yaitu titik awal dan titik akhir. Lintasan ini merepresentasikan *geodesic distance* antara kedua titik tersebut. Karena *geodesic path* merupakan representasi dari *geodesic distance*, maka sebelum melakukan proses pembuatan *geodesic path,* perlu dilakukan proses *Fast Marching Method on Triangulated Domain* untuk menghitung *geodesic distance* dari titik awal ke semua titik lainnya.

Inti dari pembuatan *geodesic path* adalah melakukan propagasi balik *(back propagation)* dimulai dari titik akhir sepanjang penurunan gradien dari *distance map* sampai diperoleh titik awal. *Distance Map* adalah matriks berukuran *n* yang

berisi *geodesic distance* tiap titik dari titik awal. *n* adalah jumlah titik. Jadi dengan kata lain pembuatan *geodesic path* adalah memecahkan *Ordinary Differential Equation (ODE)* :

$$\frac{dX(s)}{ds} = -\overrightarrow{\nabla T} \qquad 2.12$$

$$X(0) = v_0$$

dimana $X(s)$ adalah fungsi kurva parametrik yang merepresentasikan lintasan *geodesic path* pada permukaan objek 3 dimensi dan v_0 adalah titik awal. Ada 2 hal yang perlu diperhatikan dalam melakukan pembuatan *geodesic path* yaitu : *Current Vertex* dan *Current Face*. *Current Vertex* adalah titik yang akan terus dipropagasi. Setiap kali bergerak, posisi *Current Vertex* tersebut disimpan, karena posisi-posisi inilah yang nantinya membentuk *geodesic path*. Sedangkan *Current Face* adalah segitiga dimana *Current Vertex* akan dipropagasi.

Berikut langkah-langkah untuk pembuatan *geodesic path* :

- Tetapkan *Current Vertex* sebagai *vertex* yang akan dipropagasi. Sebagai inisialisasi awal *Current Vertex* adalah titik akhir.
- Lakukan Prosedur Khusus untuk memilih *Current Face* yaitu segitiga dimana *Current Vertex* akan dipropagasi.
- Lakukan Prosedur Umum untuk propagasi *Current Vertex* pada *Current Face* sampai *Current Vertex* di tepi *Current Face*, atau dengan kata lain sampai *Current Vertex* hampir keluar dari *Current Face*.

- Jika setelah dipropagasi *Current Vertex* berada di tengah *edge*, maka *Current Face* di set segitiga yang merupakan tetangga dari *Current Face* sebelumnya melalui *edge* yang memuat *Current Vertex* dan lakukan kembali Prosedur Umum dengan *Current Face* yang baru.
- Jika setelah dipropagasi *Current Vertex* berada tepat pada sebuah *vertex*, maka lakukan Prosedur Khusus terlebih dahulu sebelum Prosedur Umum.
- Prosedur Umum dan Prosedur Khusus di atas terus dilakukan sampai *Current Vertex* mencapai titik awal.

2.3.1 Prosedur Khusus

Tujuan dari prosedur ini adalah memilih segitiga yang memuat *Current Vertex* sebagai segitiga di mana *Current Vertex* akan dipropagasikan. Langkah-langkah Prosedur Khusus adalah sebagai berikut :

- Pada *Current Vertex* pilih *vertex* tetangga dengan nilai T yang terkecil untuk menjadi *Vertex* acuan.
- Pilih segitiga yang memuat *Current Vertex* dan *Vertex* acuan, dimana *vertex* lain pada segitiga tersebut mempunyai nilai T terkecil dibandingkan pada segitiga lainnya.
- Set *Current Face* dengan segitiga yang dipilih tersebut.

2.3.2 Prosedur Umum

Prosedur ini adalah inti dari proses pembuatan *geodesic path* karena pada prosedur ini dilakukan proses propagasi *Current Vertex*. Prosedur ini terdiri dari 2 proses utama yaitu :

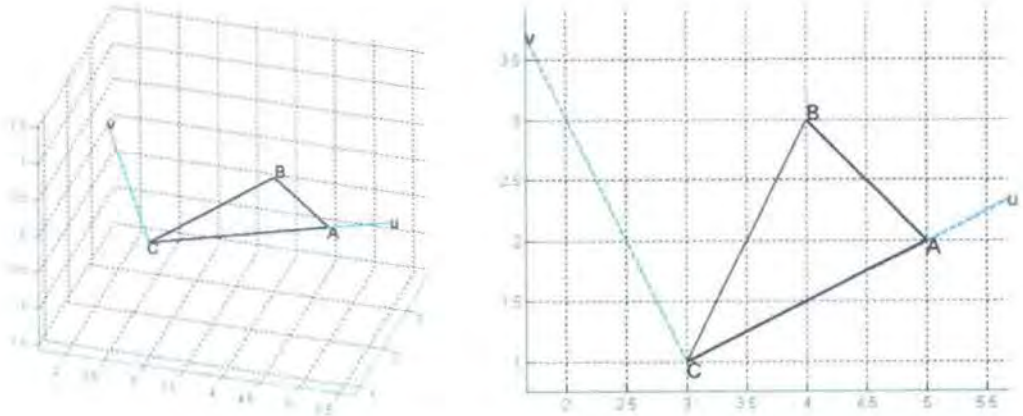
- Inisialisasi Triangular Interpolation Quadratic pada *Current Face.*
- Propagasi Balik *Current Vertex* pada *Current Face.*

2.3.2.1 Inisialisasi Triangular Interpolation Quadratic

Tujuan dari langkah ini ialah untuk mencari fungsi T yang merupakan fungsi jarak suatu titik terhadap titik awal. Langkah Inisialisasi *Triangular Interpolation Quadratic* ini dilakukan di tiap *face* segitiga yang dilewati ketika dilakukan *back propagation* dari titik akhir sampai ke titik awal.

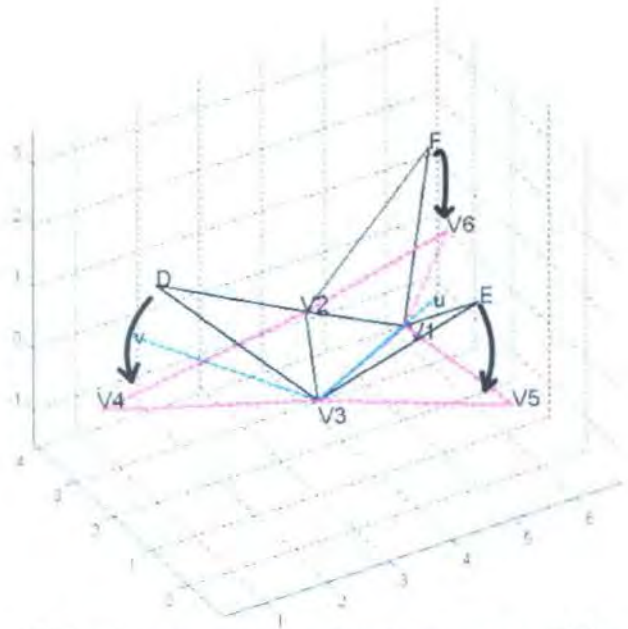
Misalkan titik yang akan dipropagasi adalah titik P dan segitiga yang memuat titik P tersebut adalah ΔABC, lalu dari ΔABC tersebut dibentuk basis ortonormal R^2 dengan titik C sebagai pusat basis. Vektor basis u diperoleh dari normalisasi vektor CA, yaitu dengan rumus : $u = \overrightarrow{CA} / length(\overrightarrow{CA})$ dan Vektor basis v diperoleh melalui rumus : $v = normalize(u \times \overrightarrow{CB} \times u)$.

Pada gambar 2-12 tampak ortonormal basis R^2 dengan titik C sebagai pusat basis, vektor u dan vektor v sebagai vektor basis yang saling tegak lurus.

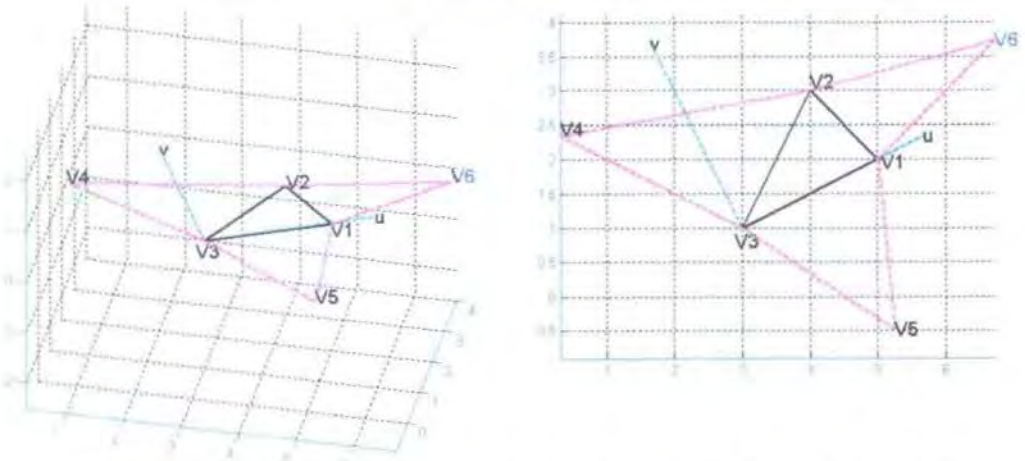


Gambar 2-12 Ortonormal basis R^2 pada ΔABC

Setelah basis ortonormal dibentuk, dilakukan proses *unfolding triangles* dari ketiga tetangga dari ΔABC agar menjadi sebidang dengan ΔABC sehingga diperoleh 6 buah titik $V_i(x_i,y_i)$, $i=\{1,2,3,4,5,6\}$ pada ortonormal basis R^2 yang telah dibuat sebelumnya. Ketiga titik pertama mengacu pada titik A,B,C dan ketiga titik selanjutnya mengacu pada titik-titik dari 3 segitiga yang merupakan tetangga dari ΔABC.



Gambar 2-13 Pendataran segitiga-segitiga tetangga ke basisR^2 pada ΔABC



Gambar 2-14 Hasil pendataran segitiga-segitiga tetangga ke basisR^2 pada ΔABC

Dari enam buah titik $V_i(x_i,y_i)$, $i=\{1,2,3,4,5,6\}$, tetapkan nilai fungsi T(x,y) dengan nilai *geodesic distance* (nilai T_i) tiap titik tersebut yang telah dihasilkan dari proses *Fast Marching Method on Triangulated Domain*. Karena interpolasi yang digunakan adalah kuadratik maka fungsi yang dibangun berbentuk :

$$T(x,y) = a + bx + cy + dxy + ex^2 + fy^2 \qquad 2.13$$

Dengan memasangkan tiap titik $V_i(x_i,y_i)$ dan T_i maka diperoleh 6 persamaan dengan 6 variabel yang tidak diketahui (a,b,c,d,e,f). Dalam persamaan matriks dapat dinyatakan melalui persamaan :

$$\begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \end{bmatrix} = \begin{bmatrix} 1 & x_1 & y_1 & x_1y_1 & x_1^2 & y_1^2 \\ 1 & x_2 & y_2 & x_2y_2 & x_2^2 & y_2^2 \\ 1 & x_3 & y_3 & x_3y_3 & x_3^2 & y_3^2 \\ 1 & x_4 & y_4 & x_4y_4 & x_4^2 & y_4^2 \\ 1 & x_5 & y_5 & x_5y_5 & x_5^2 & y_5^2 \\ 1 & x_6 & y_6 & x_6y_6 & x_6^2 & y_6^2 \end{bmatrix} \times \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix}$$

$$T = MC \qquad 2.14$$

Matriks C dalam persamaan 2.14 di atas dapat dicari dengan menggunakan metode *Lower-Upper Decompotition* atau metode *Gauss-Jordan Elimination.*

Dengan menggunakan persamaan permukaan tersebut, maka fungsi vektor gradient dari setiap titik di dalam ortonormal basis ΔABC dapat dihitung dengan :

$$G(x,y) = \nabla T(x,y) = \left\{ \frac{\partial T(x,y)}{\partial x}, \frac{\partial T(x,y)}{\partial y} \right\} = \{b + dy + 2ex, c + dx + 2fy\} \qquad 2.15$$

Vektor gradien ini kemudian dapat digunakan untuk mempropagasi titik P di dalam ΔABC sampai titik P tersebut keluar dari ΔABC.

2.3.2.2 Back Propagation Current Vertex pada Current Face

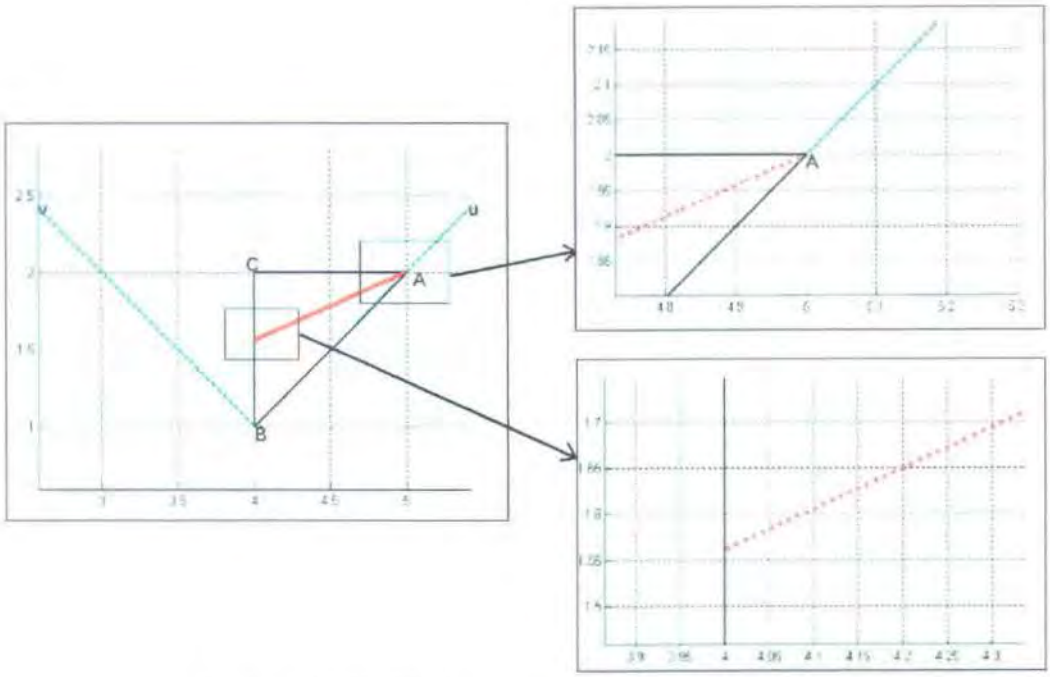
Proses ini bertujuan untuk memprediksi lintasan *geodesic path* pada *current face*. Suatu pendekatan numerik yang dilakukan untuk memprediksi lintasan tersebut adalah dengan mempropagasi mundur *current vertex* sampai titik tersebut keluar dari *current face*.

Pendekatan Numerik tersebut dapat dirumuskan sebagai :

$$P' = P - \delta \times \nabla T(P) \qquad 2.16$$

dengan P adalah *current vertex* dan δ adalah konstanta skalar yang menunjukkan besar step pengurangan dari kurva. Setiap titik P yang diperoleh disimpan ke dalam suatu *list point* pembentuk *geodesic path.*

Pada gambar 2-15 (kiri) tampak hasil *back propagation* dengan *current vertex* asal yaitu titik A dan *current face* adalah ΔABC. Garis merah dari A yang memotong ΔABC menunjukkan lintasan hasil propagasi titik A, sedangkan pada gambar 2-15 (kanan atas dan kanan bawah) tampak inset dengan perbesaran 5 kali dari gambar 2-15 (kiri). Pada gambar ini tampak titik-titik yang merupakan hasil propagasi titik A menggunakan rumus persamaan 2.16 dengan $\delta = 0.01$.



Gambar 2-15 Hasil back propagation dari titik A pada ΔABC

2.3.3 Koordinat Barycentric

Koordinat Barycentric ini pertama kali ditemukan oleh Möbius pada tahun 1827. Pada sebuah segitiga yang dibentuk oleh 3 titik (V_1, V_2, V_3), suatu titik P di dalam sebuah segitiga dapat dinyatakan secara unik dengan koordinat Barycentric (a_1, a_2, a_3).

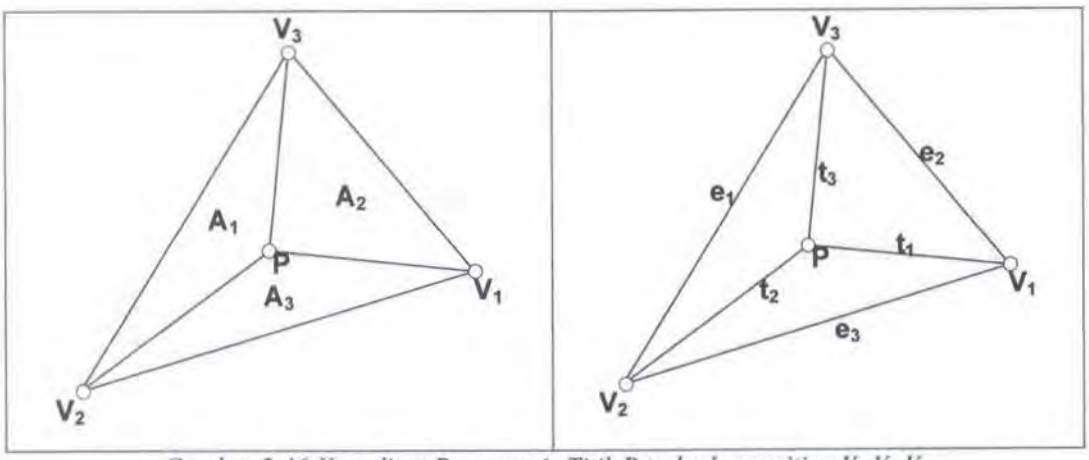
$$P = a_1V_1 + a_2V_2 + a_3V_3 \qquad 2.17$$

dimana : $a_1 + a_2 + a_3 = 1$ dan $0 \le a_1, a_2, a_3 \le 1$

Karena $a_1 + a_2 + a_3 = 1$, maka $a_3 = 1 - a_1 - a_2$. Sehingga persamaan 2.17 dapat disederhanakan menjadi :

$$P = a_1V_1 + a_2V_2 + (1 - a_1 - a_2)V_3$$

$$P = a_1(V_1 - V_3) + a_2(V_2 - V_3) + V_3 \qquad 2.18$$



Gambar 2-16 Koordinat Barycentric Titik P terhadap segitiga $V_1 V_2 V_3$

Nilai koordinat Barycentric a_1, a_2, dan a_3 menyatakan fraksi luas segitiga-segitiga pembentuk segitiga $V_1V_2V_3$. Pada gambar 2-16 (kiri) ditunjukkan bahwa A_1 adalah luas dari ΔPV_2V_3, A_2 adalah luas dari ΔPV_1V_3 dan A_3 adalah luas dari ΔPV_1V_2. Nilai A_1, A_2, A_3 dapat dicari dengan menggunakan rumus luas segitiga jika diketahui panjang ketiga sisinya (lihat gambar 2-16 kanan).

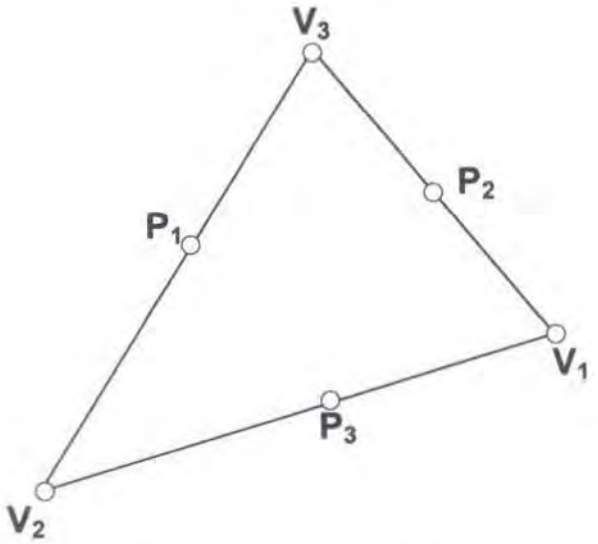
$$A_1 = \sqrt{s(s-e_1)(s-t_2)(s-t_3)}\text{, dengan } s = (e_1 + t_2 + t_3)/2$$
$$A_2 = \sqrt{s(s-e_2)(s-t_1)(s-t_3)}\text{, dengan } s = (e_2 + t_1 + t_3)/2 \qquad 2.19$$
$$A_3 = \sqrt{s(s-e_3)(s-t_1)(s-t_2)}\text{, dengan } s = (e_3 + t_1 + t_2)/2$$

Setelah nilai A_1, A_2, dan A_3 telah diperoleh, maka nilai a_1, a_2, dan a_3 dihitung dengan :

$$a_1 = \frac{A_1}{A_1 + A_2 + A_3},\ a_2 = \frac{A_2}{A_1 + A_2 + A_3},\ a_3 = \frac{A_3}{A_1 + A_2 + A_3} \qquad 2.20$$

Dalam pemakaiannya, koordinat Barysentric ini cukup istimewa karena memiliki kondisi khusus yaitu :

- Jika salah satu dari nilai a_1, a_2, atau a_3 bernilai 0, maka titik P berada tepat pada salah satu sisi segitiga $V_1V_2V_3$.

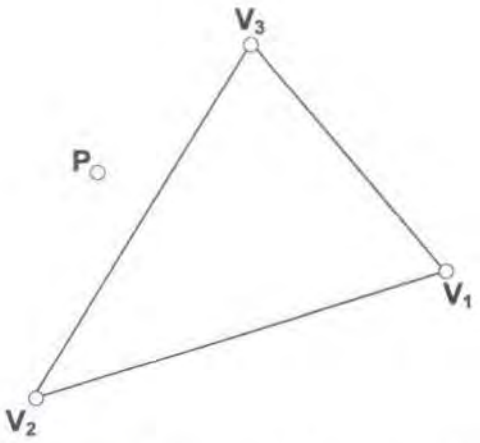


Gambar 2-17 Koordinat Barycentric Titik P untuk kondisi khusus pertama

Pada gambar 2-17, jika a_1 yang bernilai 0 maka titik P akan berada pada edge yang menghubungkan V_2V_3 (contoh : titik P_1). Jika a_2 yang bernilai 0 maka titik P akan berada pada edge yang menghubungkan V_1V_3 (contoh : titik P_1). Jika a_3 yang bernilai 0 maka titik P akan berada pada edge yang menghubungkan V_1V_2 (contoh : titik P_3).

- Jika salah satu dari nilai a_1, a_2, atau a_3 lebih besar dari 1 atau kurang dari 0, maka titik P berada diluar segitiga $V_1V_2V_3$.

  Sebagai contoh, pada gambar 2-18 tampak titik P berada diluar segitiga $V_1V_2V_3$. Hal ini dikarenakan karena nilai a_1 bernilai *-0.25*.



Gambar 2-18 Koordinat Barycentric Titik P untuk kondisi khusus kedua

Karena keistimewaannya, koordinat Barycentric ini sering digunakan pada bidang grafika terutama pada persoalan Ray Tracing dan Ray Casting. Di dalam proses pembuatan Geodesic Path, koordinat Barycentric ini digunakan untuk menyimpan hasil proses *back propagation* sebuah titik (Current Vertex) pada sebuah segitiga (Current Face) seperti yang telah dijelaskan pada sub-bab 2.3.2.2. Selain digunakan untuk penyimpanan, koordinat Barycentric ini juga berguna untuk menentukan apakah proses propagation dari current vertex itu telah berakhir atau belum. Proses propagation berakhir apabila current vertex telah keluar dari segitiga.

2.4. HEAP

Struktur data ***(binary) heap*** adalah sebuah objek array yang dapat ditampilkan sebagai sebuah *nearly complete binary tree*. Masing-masing node dari *tree* berkorespondensi pada sebuah elemen dari array yang menyimpan nilai dari node. *Tree* diisi secara lengkap pada semua level kecuali kemungkinan tree di level *(depth)* paling bawah, yang diisikan dari paling kiri sampai ke sebuah node.

Sebuah array *A* yang merepresentasikan sebuah heap adalah memiliki dua atribut: *length*[*A*], jumlah dari elemen dalam array *A*, dan *heap-size*[*A*], jumlah dari elemen dalam heap disimpan dalam array *A*. Jika semua elemen dalam heap disimpan dalam array maka *length*[*A*] = *heap-size*[*A*].

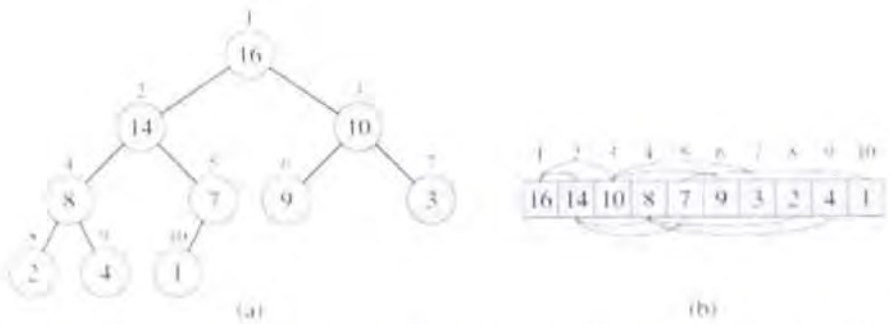
Pada sebuah tree, didefinisikan bahwa akar *(root)* dari tree adalah elemen pertama atau elemen paling atas dari tree tersebut. Jika diberikan indeks *i* dari sebuah node, maka indeks dari parent-nya dinyatakan sebagai PARENT(*i*), indeks dari anak kiri dinyatakan LEFT(*i*), dan indeks dari anak kanan dinyatakan RIGHT(*i*). Ketiganya dapat dihitung hanya sebagai berikut :

```
PARENT(i)
   return i/2

LEFT(i)
   return 2i

RIGHT(i)
   return 2i + 1
```

Pada kebanyakan komputer, prosedur LEFT dapat menghitung $2i$ dalam satu instruksi sederhana dengan menggeser representasi biner dari *i* ke kiri satu bit. Dengan cara yang sama, prosedur RIGHT dapat menghitung $2i + 1$ secara cepat dengan menggeser representasi biner dari *i* ke kiri satu bit dan menambahkan 1 sebagai *low-order bit*. Prosedur PARENT dapat menghitung $i/2$ dengan menggeser posisi representasi biner dari *i* ke kanan satu bit.

Gambar 2-19 menunjukkan sebuah heap yang dapat dinyatakan sebagai *binary tree* dan sebuah *array*. Bilangan di dalam lingkaran pada masing-masing

node dalam *tree* adalah nilai yang disimpan pada node itu. Bilangan di atas sebuah node adalah index yang berkorespondensi dalam array. Pada array kurva di atas dan di bawah array adalah garis-garis yang menunjukkan relasi *parent-child*; *parents* selalu berada di kiri anak-anaknya *(child)*.



Gambar 2-19 Sebuah heap yang ditunjukkan sebagai (a) binary tree dan (b) sebuah array

Ada dua jenis *binary heap tree* : max-heap dan min-heap. Pada kedua jenis tersebut, nilai dalam node harus memenuhi sebuah *heap property*, spesifikasi yang bergantung pada jenis heap. Dalam sebuah max-heap, *max-heap property* adalah setiap node *i* selain root (node paling atas), memenuhi :

$$A[PARENT(i)] \geq A[i]$$

Dengan kata lain setiap node nilainya lebih kecil atau sama dengan nilai node *parent*-nya. Dengan demikian, elemen paling besar dalam sebuah max-heap disimpan dalam root, dan subtree dari sebuah node berisi nilai tidak lebih besar daripada isi dari node itu sendiri. Sebuah min-heap diatur dengan cara yang berlawanan dengan max-heap; *min-heap property* adalah setiap node *i* selain root, memenuhi :

$$A[PARENT(i)] \leq A[i]$$

Sehingga elemen terkecil dalam sebuah min-heap terletak pada root.

Dengan adanya property tersebut, struktur data Heap dapat diaplikasikan sebagai metode untuk pengurutan yaitu metode Heapsort. Selain untuk metode Heapsort, struktur data Heap juga dapat diaplikasikan untuk antrian prioritas *(priority queue)*. Priority queue adalah struktur data queue yang diatur sedemikian rupa sehingga elemen pertama *(head)* dari queue tersebut adalah elemen yang memiliki prioritas utama. Sama seperti *binary heap tree*, priority queue juga mempunyai 2 jenis yaitu : *Max-priority queue* dan *Min-priority queue*. Pada Max-priority queue, elemen pertama adalah elemen yang selalu memiliki nilai terbesar dari elemen lainnya. Sebaliknya, pada Min-priority queue, elemen pertama adalah elemen yang selalu memiliki nilai terkecil dari elemen lainnya.

Dalam Tugas Akhir ini digunakan struktur data min-priority queue untuk menyimpan titik-titik yang menjadi anggota himpunan *Closed Vertex* atau dengan kata lain titik-titik yang statusnya adalah *close*. Setiap melakukan tahap perulangan (sub bab 2.2.3) akan diambil titik *trial* yang merupakan titik pada himpunan *closed vertex* dengan nilai T terkecil / minimum. Jika nilai yang dibandingkan dalam heap adalah nilai T dari tiap titik, maka titik *trial* adalah titik yang berada pada *head* dari min-priority queue tersebut. Berikut akan dijelaskan lebih lanjut mengenai struktur data min-priority queue.

2.4.1 Min-Priority Queue

Min-Priority Queue adalah struktur data queue yang elemen pertamanya *(head)* selalu merupakan elemen dengan nilai *(key)* terkecil. Min-priority queue ini adalah bentuk array dari min-heap binary tree. Sehingga Min-priority queue

juga harus memenuhi min-heap property. Dalam penerapannya, min-priority queue memiliki beberapa prosedur sebagai berikut :

- PUSH-HEAP(A, x), yaitu memasukkan elemen x ke dalam himpunan array A. Operasi ini dapat ditulis sebagai $A \leftarrow A \cup \{x\}$.
- POP-HEAP(A), yaitu mengeluarkan elemen dari A dengan nilai *(key)* terkecil.
- DECREASE-KEY(A, i, k), yaitu menurunkan nilai dari elemen ke-i dari A dengan nilai baru k yang lebih kecil dari nilai sebelumnya.

Pada algoritme FMM on TD, prosedur PUSH-HEAP ini dijalankan ketika memasukkan suatu vertex baru ke dalam himpunan Closed Vertex. Sedangkan fungsi POP-HEAP dijalankan waktu mengambil Trial Vertex yang merupakan titik dari himpunan Closed Vertex dengan nilai T terkecil. Dan prosedur DECREASE-KEY dijalankan ketika meng-update nilai T pada suatu Vertex yang statusnya close dengan nilai T baru yang lebih kecil.

Selain prosedur di atas, min-priority queue juga mempunyai beberapa prosedur dasar yang digunakan untuk memelihara min-heap property. Prosedur-prosedur tersebut antara lain :

- MIN-HEAPIFY(A, i)
- UP-HEAP(A, i)

Penjelasan dari prosedur-prosedur tersebut ada pada sub bab berikut.

2.4.2 Memelihara Min-Heap Property pada Min-Priority Queue

MIN-HEAPIFY adalah *subroutine* penting untuk memelihara Min-Heap Property. Masukan dari prosedur ini adalah sebuah array A dan index i dari array. Ketika MIN-HEAPIFY dipanggil, diasumsikan bahwa *sub-trees* yang akarnya pada LEFT(i) dan RIGHT(i) adalah juga merupakan min-heap, tetapi pada kenyataannya $A[i]$ mungkin lebih besar daripada anak-anaknya, hal akan melanggar properti min-heap. Untuk itu tujuan dari prosedur MIN-HEAPIFY adalah untuk membiarkan nilai pada $A[i]$ bergerak turun *(float down)* dalam tree sehingga *sub-tree* yang akarnya pada index i juga menjadi sebuah min-heap.

```
MIN-HEAPIFY(A, i)
 1 l ← LEFT(i)
 2 r ← RIGHT(i)
 3 if l ≤ heap-size[A] and A[l] < A[i]
 4    then smallest ← l
 5    else smallest ← i
 6 if r ≤ heap-size[A] and A[r] < A[smallest]
 7    then smallest ← r
 8 if smallest ≠ i
 9    then  exchange A[i] ↔ A[smallest]
10          MIN-HEAPIFY(A, smallest)
```

Pada prosedur MIN-HEAPIFY, elemen-elemen $A[i]$, A[LEFT(i)], dan A[RIGHT(i)] ditentukan mana yang paling kecil, dan index dari elemen terkecil tersebut disimpan dalam variabel *smallest*. Jika $A[i]$ adalah elemen yang paling kecil, maka subtree yang akarnya pada node-i adalah sebuah min-heap dan akhiri prosedur MIN-HEAPIFY. Sebaliknya, jika salah satu dari kedua anak memiliki elemen terkecil, maka $A[i]$ ditukar dengan A[*smallest*], yang menyebabkan node-i dan anaknya memenuhi properti min-heap. Node yang berindeks *smallest*, bagaimanapun juga sekarang memiliki nilai asal $A[i]$, dan dengan demikian subtree memiliki akar pada smallest ini dapat melanggar properti min-heap.

Konsekuensinya, MIN-HEAPIFY harus dipanggil secara rekursif pada subtree tersebut.

Pada penerapan min-priority queue, prosedur MIN-HEAPIFY ini dipanggil ketika menjalankan fungsi POP-HEAP.

```
POP-HEAP(A)
 1 if heap-size[A] < 1
 2    then error "heap underflow"
 3 min ← A[1]
 4 A[1] ← A[heap-size[A]]
 5 heap-size[A] ← heap-size[A] - 1
 6 MIN-HEAPIFY(A, 1)
 7 return min
```

Pada fungsi POP-HEAP setelah nilai pada $A[i]$ diambil, maka nilai pada $A[i]$ diganti dengan nilai A[heap-size[A]] yang merupakan nilai terbesar pada A. Langkah ini mengakibatkan properti min-heap dilanggar. Untuk itu perlu dilakukan prosedur MIN-HEAPIFY untuk menjaga properti min-heap.

Selain prosedur MIN-HEAPIFY, digunakan juga prosedur UP-HEAP yang cara kerjanya mirip dengan prosedur MIN-HEAPIFY. Bedanya adalah prosedur ini bertujuan untuk membiarkan nilai $A[i]$ bergerak naik dalam tree selama nilai $A[i]$ lebih kecil dari nilai A[PARENT(i)].

```
UP-HEAP(A, i)
 1 while i > 1 and A[PARRENT(i)] > A[i]
 2    do exchange A[PARRENT(i)] ↔ A[i]
 3    i ← PARRENT(i)
```

Pada prosedur UP-HEAP, elemen $A[i]$ dan A[PARRENT(i)] dibandingkan jika nilai $A[i]$ lebih besar maka sudah memenuhi property Min-Heap dan akhiri prosedur UP-HEAP. Sebaliknya, jika $A[i]$ lebih kecil maka nilai $A[i]$ ditukar dengan A[PARRENT(i)], bagaimanapun juga setelah pertukaran tersebut nilai

A[PARRENT(*i*)] yang baru dapat melanggar properti Min-Heap. Untuk itu proses pertukaran antara elemen *A*[PARRENT(*i*)] dan *A*[*i*] dilakukan terus menerus selama A[*i*] lebih kecil dari *A*[PARRENT(*i*)].

Pada penerapan min-priority queue, prosedur UP-HEAP ini dipanggil ketika menjalankan prosedur PUSH-HEAP.

```
PUSH-HEAP(A, x)
 1 heap-size[A] ← heap-size[A] + 1
 2 A[heap-size[A]] ← x
 3 UP-HEAP(A, heap-size[A])
```

Pada prosedur PUSH-HEAP, nilai baru *x* dimasukkan pada array *A* urutan paling belakang. Dengan nilai yang baru ini, min-heap property dapat dilanggar, untuk itu perlu dilakukan UP-HEAP pada elemen yang baru tersebut.

Selain pada prosedur PUSH-HEAP, prosedur UP-HEAP juga dipanggil ketika menjalankan prosedur DECREASE-KEY.

```
DECREASE-KEY(A, i, k)
 1 if k > A[i]
 2    then error "new key is larger then current key"
 3 A[i] ← k
 4 UP-HEAP(A,i)
```

Pada prosedur DECREASE-KEY, setelah nilai pada elemen *A*[*i*] berhasil diganti dengan nilai *k* yang lebih kecil. Dengan nilai yang baru ini, min-heap property dapat dilanggar, untuk itu perlu dilakukan UP-HEAP pada elemen tersebut.

2.4.3 Kompleksitas Min-Prioriy Queue

Untuk menghitung kompleksitas waktu pada struktur data Min-Priority Queue perlu dilakukan analisis algoritme terhadap prosedur-prosedur untuk memelihara Min-Heap Property. Prosedur-prosedur tersebut antara lain prosedur MIN-HEAPIFY dan UP-HEAP. Berikut analisis algoritme pada prosedur MIN-HEAPIFY :

```
MIN-HEAPIFY(A, i)                                   cost    times
 1 l ← LEFT(i)                                      c1      1
 2 r ← RIGHT(i)                                     c2      1
 3 if l ≤ heap-size[A] and A[l] < A[i]              c3      1
 4    then smallest ← l                             c4      1
 5    else smallest ← I                             c5      1
 6 if r ≤ heap-size[A] and A[r] < A[smallest]       c6      1
 7    then smallest ← r                             c7      1
 8 if smallest ≠ I                                  c8      1
 9    then  exchange A[i] ↔ A[smallest]             c9      1
10       MIN-HEAPIFY(A, smallest)                   c10     2n/3
```

MIN-HEAPIFY dijalankan secara rekursif pada subtree yang memiliki root pada salah satu anak dari A[i]. Pada *worst case*, besar dari subtree tersebut adalah $2n/3$, yang akan terjadi ketika baris terakhir dari tree tepat setengah penuh. Dari analisis di atas diperoleh :

$$T(n) = \begin{cases} c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7 + c_8 + c_9 & , jika \quad n = 1 \\ (c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7 + c_8 + c_9) + c_{10} T(2n/3) & , jika \quad n > 1 \end{cases} \qquad 2.21$$

Jika nilai masing-masing *cost* dianggap sebagai konstanta, maka diperoleh persamaan recurrence sebagai berikut :

$$T(n) = \begin{cases} \Theta(1) & , jika \quad n = 1 \\ T(2n/3) + \Theta(1) & , jika \quad n > 1 \end{cases} \qquad 2.22$$

Dengan menggunakan Master Theorem : $T(n) = aT(n/b) + f(n)$, maka diperoleh nilai a = 1, b = 3/2, dan $f(n)$ = $\Theta(1)$. Pada Case 2 dari Master Theorem disebutkan bahwa jika $f(n) = \Theta(n^{\log_b a})$, maka $T(n) = \Theta(n^{\log_b a} \lg n)$. Maka dengan nilai $\log_b a = 0$, diperoleh $T(n) = \Theta(\lg n)$. Sehingga dapat disimpulkan bahwa kompleksitas waktu dari prosedur MIN-HEAPIFY adalah $O(\lg n)$ dimana n adalah ukuran dari heap atau tree.

Sedangkan untuk analisis prosedur UP-HEAP adalah sebagai berikut :

```
UP-HEAP(A, i)                                   cost    times
1 while i > 1 and A[PARRENT(i)] > A[i]          c1      h
2    do exchange A[PARRENT(i)] ↔ A[i]           c2      h - 1
3       i ← PARRENT(i)                          c3      h - 1
```

Pada prosedur UP-HEAP terdapat sebuah *looping*. Pada *worst case*, perintah di dalam *looping* ini dilakukan sebanyak h-1 kali. Nilai h menunjukkan *height* dari tree. Nilai h dapat dinyatakan : $h = \lfloor \log_2 n \rfloor + 1$. Dari analisis di atas diperoleh :

$$T(n) = c_1 h + c_2(h-1) + c_3(h-1)$$

$$T(n) = (c_1 + c_2 + c_3)h + (-c_2 - c_3)$$

$$T(n) = (c_1 + c_2 + c_3)(\log_2(n) + 1) + (-c_2 - c_3)$$

$$T(n) = (c_1 + c_2 + c_3)\log_2(n) + c_1$$

$$T(n) = \Theta(\log_2(n)) \in \Theta(\lg n) \qquad 2.23$$

Sehingga dapat disimpulkan bahwa kompleksitas waktu dari prosedur UP-HEAP adalah $O(\lg n)$ dimana n adalah ukuran dari heap atau tree.

Prosedur POP-HEAP menjalankan prosedur MIN-HEAPIFY sekali, sehingga kompleksitas waktu dari prosedur POP-HEAP sama dengan kompleksitas waktu dari prosedur MIN-HEAPIFY yaitu $O(\lg n)$. Sedangkan prosedur PUSH-HEAP dan DECREASE-KEY menjalankan prosedur UP-HEAP sekali, sehingga kompleksitas waktu dari masing-masing prosedur PUSH-HEAP dan DECREASE-KEY sama dengan kompleksitas waktu dari prosedur UP-HEAP yaitu $O(\lg n)$.

Karena semua prosedur yang digunakan dalam struktur data Min-Priority Queue memiliki kompleksitas $O(\lg n)$, maka struktur data Min-Priority Queue berjalan pada kompleksitas $O(\lg n)$, dimana n adalah jumlah elemen dalam queue.

2.5. Format File Ply

Setiap orang yang baru bekerja di bidang grafika komputer (computer graphic) kadang-kadang akan mengalami kebingungan dalam menentukan format penyimpanan objek grafik. Kadang setiap programmer membuat file format sendiri untuk proyeknya, dan ini menyebabkan data objek tersebut tidak fleksibel. Diperlukan suatu format sehingga data objek bisa digunakan pihak lain.

Terdapat beberapa file format yang digunakan untuk menyimpan objek grafik dan salah satunya adalah PLY Polygon file format. Format PLY merupakan file format untuk menyimpan objek grafik yang sering digunakan untuk kepentingan pendidikan / penelitian di bidang grafika komputer. Data *Polygon*

mesh dari *Stanford University* untuk model hasil *3DScanning* juga di*release* menggunakan format file ini. Masih banyak format file yang lain seperti VRML file, poly file, obj file.

Format file PLY menggambarkan sebuah objek sebagai kumpulan dari elemen diantaranya elemen titik, permukaan/bidang dan elemen-elemen yang lainnya. Setiap tipe elemen memiliki satu atau lebih properti yang menjelaskan elemen itu sendiri seperti warna, vektor normal bidang, kordinat dari tekstur, sifat tembus(transparancy) maupun material dari elemen tersebut. Setiap file PLY digunakan hanya untuk satu buah objek grafik. Sumber data objek grafik tersebut bisa berasal dari digitasi objek, objek poligon yang berasal dari program pemodelan, range data hasil dari pemindaian objek asli dengan laser scanner atau terrain data. Namun format PLY tidak menyertakan matrik transformasi , istantiasi objek, hierarki objek seperti pada format VRML.

Objek PLY secara sederhana didefinisikan sebagai sebuah daftar (list) dari titik/vertex (x,y,z) dan daftar dari bidang/poligon(face) yang dideskripsikan sebagai index dari daftar titik(list of vertex). Bidang / poligon tersebut bisa berupa segitiga, segiempat dan seterusnya segibanyak. Kebanyakan dari file PLY menyertakan kedua informasi inti ini. Vertex dan face merupakan 2 contoh elemen yang digunakan dalam file PLY. Setiap elemen yang terdapat dalam file PLY memiliki beberapa properti yang spesifik. Informasi minimal(typical) yang biasanya menjadi properti adalah informasi (x,y,z) sebagai pembentuk vertex dan index dari list vertex sebagai pembentuk face. Aplikasi bisa membuat properti

baru untuk setiap elemen sebagai informasi tambahan untuk setiap objek, seperti properti red, green dan blue sebagai informasi warna untuk setiap vertex elemen. Properti baru yang baru ditambahkan bisa diabaikan oleh aplikasi lama jika properti tersebut tidak dikenal. Selain properti, aplikasi juga bisa membuat Elemen baru dan properti yang mendukungnya seperti elemen Edge dengan properti index vertex dan material dengan properti ambient, diffuse dan specular. Elemen baru ini juga bisa diabaikan jika aplikasi lama tidak mengenalinya.

2.5.1 Struktur File

Struktur dari file PLY secara sederhana adalah seperti dibawah ini:

```
Header
Vertex list
Face List
(list elemen yang lainnya)
```

Header merupakan beberapa baris teks yang menjelaskan objek secara keseluruhan. Header diantaranya menjelaskan mengenai format penulisan file, apakah ditulis secara ASCII atau binary, menjelaskan mengenai tipe elemen termasuk nama dari elemen(seperti *Vertex, Face, Edge*), jumlah elemen yang terdapat dalam objek, dan beberapa properti lainnya yang menjadi informasi tambahan dari elemen. Setelah Header kemudian diikuti oleh daftar elemen untuk setiap tipe elemen seperti yang terdapat pada struktur diatas.

Dibawah ini adalah contoh file PLY dengan format ASCII yang menjelaskan tentang objek Kubus. Komentar yang berada didalam kurung bukan bagian dari file, hanya sebagai keterangan yang digunakan dalam contoh.

komentar yang merupakan bagian dari file biasanya dimulai dengan keyword "comment" .

```
ply
format ascii 1.0                    {format ascii/binary,nomor versi}
comment made by anonymous           { keyword komentar }
comment this file is a cube
element vertex 8                    {"vertex" element, jumlah 8 }
property float32 x                  {x kordinat sbg properti }
property float32 y                  {y kordinat sbg properti }
property float32 z                  {z kordinat sbg properti }
element face 6                      {"face" elemen, jumlah 6 }
property list uint8 int32 vertex_index
end_header                          { akhir dari header}
0 0 0                               { mulai vertex list}
0 0 1
0 1 1
0 1 0
1 0 0
1 0 1
1 1 1
1 1 0
4 0 1 2 3                           { mulai face list }
4 7 6 5 4
4 0 4 5 1
4 1 5 6 2
4 2 6 7 3
4 3 7 4 0
```

Gambar 2-20 Contoh File Ply

Contoh diatas memperlihatkan komponen dasar dari header. Setiap bagian dari header diakhiri dengan carriage-return ASCII string yang dimulai dengan keyword. Awal dan akhir dari header adalah ply<cr> dan end_header<cr>. Karakter ply<cr> harus menjadi 4 karakter pertama dalam file PLY. Kemudian diikuti keyword "format" untuk menentukan apakah ASCII atau binary format, yang diikuti nomor versi . keyword berikutnya kemudian menjelaskan mengenai setiap elemen yang terdapat pada file PLY dan diikuti oleh properti masing-masing.

Bentuk umum dari header elemen adalah sbb:

```
element <element-name> <number-in-file>
property <data-type> <property-name-1>
property <data-type> <property-name-2>
property <data-type> <property-name-3>
```

daftar properti yang tertulis setelah elemen mendefinisikan tipe data dari properti dan urutannya dalam setiap daftar elemen. Terdapat 3 jenis tipe data yang dimiliki properti yakni scalar, string dan list. Dibawah ini adalah daftar dari tipe data scalar:

Tabel 2-1 Tipe Data format file PLY

Name	Type	Number of bytes
int8	character	1
uint8	unsigned character	1
int16	short integer	2
uint16	unsigned short integer	2
int32	integer	4
uint32	unsigned integer	4
float32	single-precision float	4
float64	double-precision float	8

Bentuk umum dari properti yang menggunakan tipe data list adalah seperti dibawah ini:

```
property list <numerical-type> <numerical-type> <property-name>
```

Contoh dari penggunaan properti ini adalah seperti pada objek kubus diatas:

```
property list uint8 int32 vertex_index
```

Ini menjelaskan bahwa properti vertex_index mengandung sebuah unsigned char uint8 yang menentukan jumlah index vertex pembentuk poligon dan diikuti oleh daftar vertex_index dengan tipe int32.

Untuk User-defined Element, format yang dipakai sama dengan format untuk elemen *vertex*, *face* ataupun *edge*. Seperti contoh dibawah ini:

```
element material 6
property ambient_red uint8                    { ambient color }
property ambient_green uint8
property ambient_blue uint8
property ambient_coeff float32
property diffuse_red uint8                    { diffuse color }
property diffuse_green uint8
property diffuse_blue uint8
property diffuse_coeff float32
property specular_red uint8                   { specular color }
property specular_green uint8
property specular_blue uint8
property specular_coeff float32
property specular_power float32               { Phong power }
```

Untuk menggunakan elemen material ini untuk setiap vertex , pada bagian akhir properti vertex bisa ditambahkan satu buah properti:

```
property material_index int32
```

yang menunjukkan bahwa untuk setiap elemen vertex yang bersangkutan mempunyai material dengan index material_index.

BAB III
PERANCANGAN PERANGKAT LUNAK

Pada bab ini akan dibahas mengenai perancangan desain sistem perangkat lunak perhitungan *Geodesic Distance* menggunakan algoritme *Fast Marching Method on Triangulated Domain* dan pembuatan *Geodesic Path.* Pembahasan ini meliputi perancangan data berupa data masukan, data pada saat proses berjalan dan data keluaran, perancangan proses berupa perancangan proses buka data model, perancangan proses *Fast Marching Method* untuk perhitungan *geodesic distance*, perancangan proses pembuatan *geodesic path* dan proses visualisasi model. Perancangan perangkat lunak ini menggunakan pendekatan desain berorientasi objek yang direpresentasikan dengan mengunakan **UML** (*Unified Modeling Language*) dengan perangkat lunak Rasional Rose Enterprise Edition 2000e.

3.1. PERANCANGAN DATA

Perangkat lunak ini secara garis besar mengelompokkan data-data yang berkaitan dengan proses perhitungan *geodesic distance,* proses pembuatan *geodesic path,* maupun proses visualisasi menjadi tiga bagian yaitu data masukan, data proses, dan data keluaran.

3.1.1 Perancangan Data Masukan

Data masukan yang digunakan untuk perangkat lunak ini berasal dari berkas masukan dengan format `ply`. Seperti apa yang telah dijelaskan pada bab II

mengenai Format file PLY, berkas ini bisa digunakan untuk menyimpan berbagai macam atribut dari suatu model baik atribut dasar seperti titik dan poligon maupun atribut tambahan yang biasanya berhubungan dengan proses visualisasi seperti material dan warna. Namun pada perangkat lunak ini, data minimal yang dibutuhkan agar proses perhitungan *geodesic distance* dan pembuatan *geodesic path* serta visualisasi bisa dilakukan adalah data titik (*Vertex/Vertices*) dan data poligon sebagai atribut dasar pembentuk model. Format penulisan dari data model yang disimpan di berkas masukan yang bertipe teks (ASCII) adalah sebagai berikut (atribut dasar) :

```
ply    ::extension file
format ascii 1.0  ::tipe file
element vertex [jml]
property float x
property float y
property float z
element face [jml]
property list uchar int vertex_indices
end_header
[data vertex]
[data face]
```

Keterangan untuk masing-masing data adalah sebagai berikut:

1. **element** : menyatakan jenis atribut model yang disimpan , dalam hal ini adalah data titik (*vertex*) dan data poligon (*face*). Dan diteruskan dengan jumlah elemen yang disimpan dalam berkas tsb.
2. **property** : menyatakan data apa saja yang disimpan untuk setiap elemen yang bersangkutan. Jika elemen berupa vertex maka propertinya merupakan koordinat *x,y,z* dengan tipe data float. sedangkan jika elemen berupa face maka propertinya merupakan list dari index vertex.

3. [data vertex] : merupakan data vertex itu sendiri, dimana data yang disimpan tergantung dari properti elemen vertex. Jika data vertex :

```
0.2  0.34  -0.56
```

maka data $x = 0.2$, $y = 0.34$, $z = -0.56$

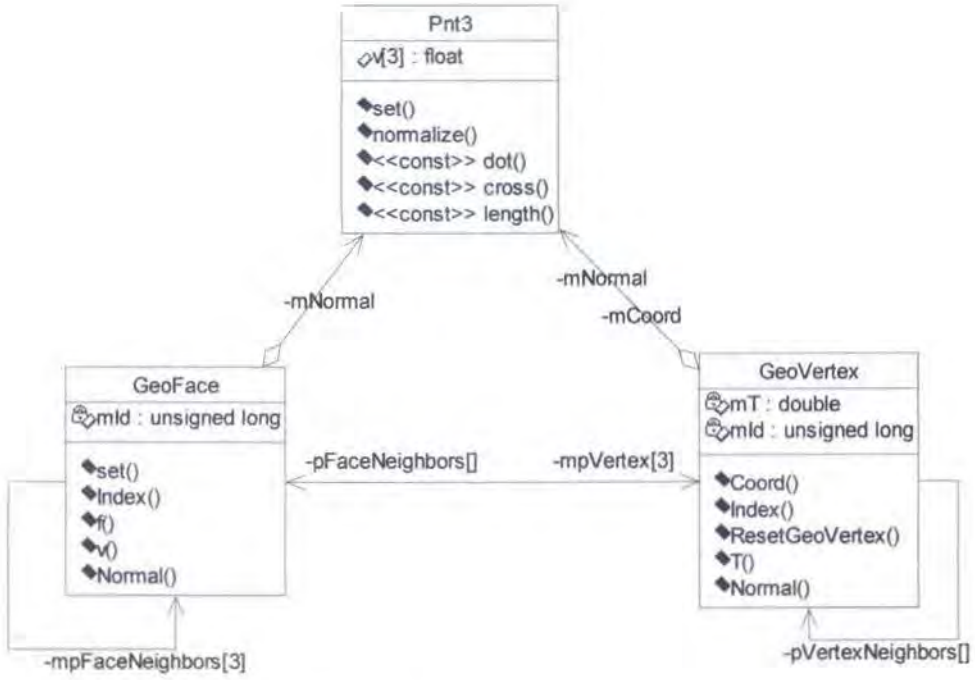
4. [data face] : merupakan data poligon dari model. Jika data face :

```
3  0  4  2
```

maka jumlah titik pembentuk poligon adalah 3. Data berikutnya (0 , 4 dan 2) adalah *index* dari *vertex* pembentuk poligon tersebut..

Proses perhitungan *geodesic distance* menggunakan algoritme *Fast Marching Method on Triangulated Domain* hanya dapat berjalan dengan masukan data poligon segitiga *(triangular mesh).* Oleh karena itu pada aplikasi ini, data masukan dibatasi harus berupa *triangular mesh.*

Untuk menampung data masukan dari berkas ply tersebut, dibuatkan *wrapper class* seperti yang tampak gambar 3.1.



Gambar 3-1 Kelas Diagram untuk data masukan

Setiap data vertex dari berkas ply akan disimpan dalam objek **GeoVertex**. GeoVertex digunakan untuk mengenkapsulasi data *vertex* yang ada pada berkas ply. Data yang dienkapsulasi diantaranya adalah data index untuk menentukan urutan dari vertex tersebut didalam array. Data index merupakan data yang sangat penting karena data face pada berkas ply dibentuk dari index ini. Setiap GeoVertex juga akan mengandung objek Pnt3 untuk menyimpan data koordinat *x*, *y* dan *z* yang berasal dari berkas ply (kelas Pnt3 akan dijelaskan kemudian) dan data vektor normal yang nantinya berguna dalam proses visualisasi model. Kedua objek Pnt3 pada GeoVertex diperlihatkan pada gambar 3.1 dengan relasi *Aggregation* antara kelas GeoVertex dengan Pnt3. Operasi yang disediakan pada GeoVertex adalah operasi-operasi untuk mengakses dan men-set nilai dari atribut yang dimilikinya.

Kelas **GeoFace** digunakan untuk mengenkapsulasi data *face* yang ada pada berkas ply. Data yang dienkapsulasi adalah data *vertex* yang membentuk *face*, dalam hal ini data yang disimpan bukan data mentahan dari berkas ply yakni 3 buah data index, melainkan pointer ke objek GeoVertex dengan index tersebut. GeoFace juga menyimpan objek Pnt3 untuk menyimpan vektor normal dari *face* tersebut dimana nantinya digunakan dalam proses visualisasi.

Kelas **Pnt3** merupakan kelas yang digunakan untuk mengenkapsulasi data yang berhubungan dengan koordinat tiga dimensi maupun vektor. Kelas ini dibuat untuk mempermudah operasi-operasi matematika maupun operasi vektor yang dikenakan pada objek ini. Operasi-operasi yang disediakan kelas Pnt3 diantaranya :

Tabel 3-1 Method pada kelas Pnt3

Method	Penjelasan
`Overloading operator (+ - * / += -= *= /= == !=)`	Digunakan untuk mempermudah operasi aritmatika yang dikenakan pada objek Pnt3
`length()`	Digunakan untuk mencari panjang dari objek Pnt3
`normalize()`	Digunakan untuk menormalisasi objek Pnt3 (mencari unit vektor)
`scale()`	Digunakan untuk menskala objek Pnt3
`dist()`	Digunakan untuk mencari jarak antara 2 objek Pnt3
`dot()`	Digunakan untuk melakukan perkalian titik (*dot product*) antara 2 objek Pnt3
`cross()`	Digunakan melakukan perkalian silang (*cross product*) antara 2 objek Pnt3

Method	Penjelasan
`normal()`	Digunakan untuk menghitung vektor normal antara 2 objek Pnt3 (vektor yang tegak lurus dengan 2 objek Pnt3)

3.1.2 Perancangan Data Proses

Data Proses adalah data yang dibutuhkan selama aplikasi berjalan. Data yang dibutuhkan diantaranya adalah data untuk melakukan proses perhitungan *geodesic distance*, proses pembuatan *geodesic path* dan data yang dibutuhkan untuk keperluan visualisasi model ke layar. Ada beberapa data utama yang berperan penting yaitu :

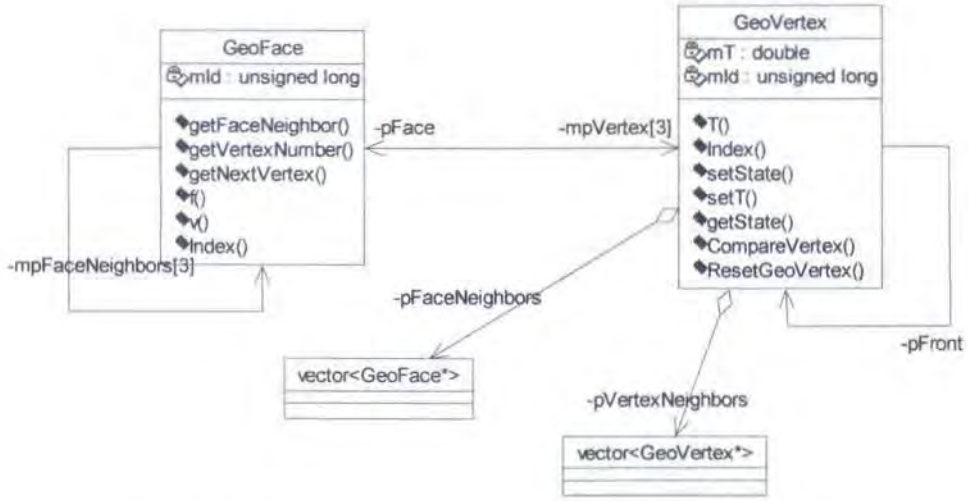
1. `Data Permukaan` : merupakan data yang diperlukan untuk melakukan proses perhitungan *geodesic distance*, pembuatan *geodesic path* maupun proses visualisasi permukaan dari model. Data ini terdiri dari :
   - `Vertex` : merupakan titik-titik kordinat pembentuk model. Vertex merupakan data yang nantinya akan diposisikan ulang (dirata-ratakan / *refinement*) untuk menghasilkan permukaan yang halus.
   - `Face` : Permukaan yang akan dibuat merupakan permukaan yang dibentuk oleh rangkaian poligon (*Polygonal Mesh*) dan tipe poligon yang digunakan adalah Segitiga (*Triangle*).
2. `Data Lintasan` : merupakan data yang diperlukan untuk melakukan proses pembuatan *geodesic path* dan visualisasi *geodesic path* tersebut pada model. Data ini terdiri dari :

- `Point` : Merupakan titik yang berada di tengah edge. Titik ini adalah awal dari lintasan pada suatu face. Edge adalah garis yang menghubungkan dua buah vertex. Setiap point mengandung informasi dua *vertex* yang membentuk edge dimana point tersebut berada, posisi point tersebut pada edge dan informasi *face* dimana point tersebut berada serta Sub Point List. Data point ini digunakan dalam membentuk *Geodesic Path.*
- `Sub Point List` : Merupakan daftar titik yang membentuk *geodesic path* pada suatu *face*. Setiap Sub Point mengandung informasi koordinat barycentric terhadap segitiga dimana Sub Point itu berada. Data ini digunakan dalam membentuk *Geodesic Path.*

3. Data Visualisasi Model merupakan data yang digunakan untuk memberikan efek pada model yang digambar di layar diantaranya data warna dari model, garis, data cahaya dan material dari permukaan model, jenis penggambaran yang diinginkan oleh pengguna.

Dibawah ini akan dibahas masing-masing struktur data yang digunakan untuk proses perhitungan *geodesic distance*, proses pembuatan *geodesic path*, dan proses visualisasi .

3.1.2.1 Data Proses Perhitungan Geodesic Distance



Gambar 3-2 Kelas Diagram untuk data proses Perhitungan Geodesic Distance

Data yang dibutuhkan untuk proses perhitungan *geodesic distance* adalah data permukaan meliputi data Vertex dan Face. Data vertex dienkapsulasi dalam kelas **GeoVertex**. Sedangkan data face dienkapsulasi dalam kelas **GeoFace**.

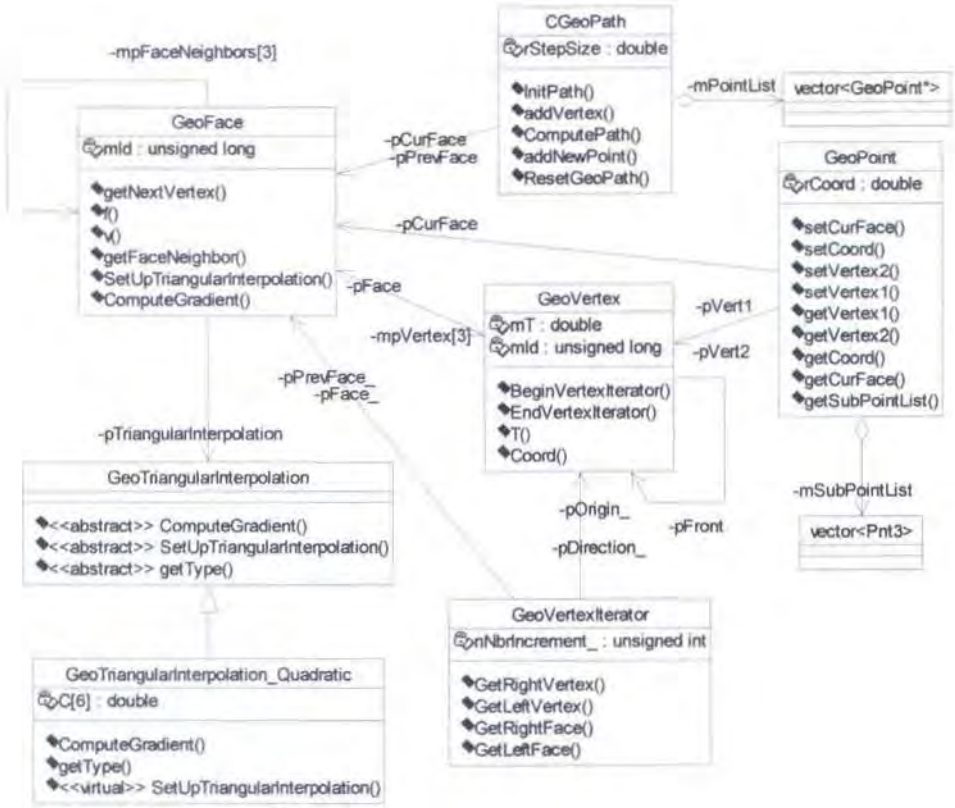
Di dalam tiap kelas GeoVertex ini terdapat informasi koordinat vertex dan variable T yang merupakan *geodesic distance* antara vertex tersebut dengan *starting vertex*. Tidak hanya itu, di dalam tiap kelas GeoVertex juga terdapat informasi berupa daftar (vector) pointer ke objek GeoVertex lainnya yang merupakan vertex tetangga dari GeoVertex tersebut. Selain itu juga terdapat daftar (vector) pointer ke objek GeoFace yang merupakan Face yang memiliki GeoVertex tersebut.

Sedangkan di dalam tiap kelas GeoFace terdapat informasi berupa 3 buah pointer ke objek GeoVertex yang membentuk segitiga (face) tersebut dan 3 buah

pointer ke objek GeoFace lainnya yang merupakan face tetangga dari GeoFace tersebut.

Selain data permukaan, proses perhitungan *geodesic distance* memerlukan 2 buah pointer ke objek GeoVertex, sebagai titik awal *(starting vertex)* dan titik akhir *(end vertex)*.

3.1.2.2 Data Proses Pembuatan Geodesic Path



Gambar 3-3 Kelas Diagram untuk data proses Pembuatan Geodesic Path

Data yang dibutuhkan dalam proses pembuatan *geodesic path* adalah data *vertex* dan *face* dari proses perhitungan *geodesic distance* dan data lintasan. Kelas **CGeoPath** digunakan untuk mengenkapsulasi data lintasan yang akan dibuat.

Di dalam kelas CGeoPath terdapat informasi berupa daftar (vector) pointer ke objek **GeoPoint.** Kelas GeoPoint ini mengenkapsulasi data titik-titik pembentuk lintasan. Kelas GeoPoint ini menyimpan SubPointList yaitu daftar (vektor) kelas Pnt3 yang merupakan koordinat barycentric dari titik pembentuk lintasan terhadap segitiga (face) dimana titik tersebut berada. Selain itu kelas GeoPoint mengandung informasi 2 buah pointer ke objek GeoVertex untuk menyimpan vertex yang membentuk edge dimana point tersebut berada dan sebuah pointer ke objek GeoFace untuk menyimpan face dimana lintasan dibuat.

Dalam melakukan proses pembuatan *geodesic path* diperlukan data untuk menyimpan fungsi T hasil interpolasi triangular kuadratik. Untuk menyimpannya digunakan kelas **GeoTriangularInterpolation_Quadratic** yang merupakan turunan dari kelas **GeoTriangularInterpolation**. Dalam kelas GeoTriangularInterpolation_Quadratic disimpan informasi 6 variabel koefisien pembentuk fungsi $T(x, y) = a + bx + cy + dxy + ex^2 + fy^2$. Selain itu juga diperlukan kelas **GeoVertexIterator** yang digunakan untuk meng-iterasi objek GeoVertex yang ada.

3.1.2.3 Data Proses Visualisasi

Data proses visualisasi dienkapusulasi pada suatu kelas yaitu kelas **StateManager**. Kelas ini digunakan untuk menyimpan data-data *environment* dari aplikasi seperti tipe penggambaran model apakah digambar dengan mode *point, line, flat polygon* atau *smooth polygon*, apakah mode *backface culling* diaktifkan atau tidak, apakah penggambaran menggunakan cahaya atau tidak,

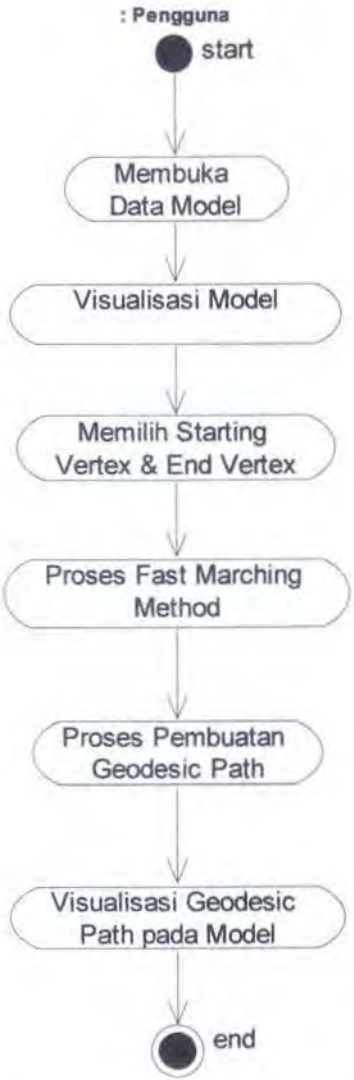
apakah nomor ID dari tiap vertex ditampilkan atau tidak, apakah panel informasi ditampilkan atau tidak. Selain itu juga digunakan untuk menyimpan warna baik warna background, warna material dari model maupun warna garis.

Gambar 3-4 Kelas StateManager

3.1.3 Perancangan Data Keluaran

Hasil pengolahan dari perangkat lunak ini adalah data nilai *geodesic distance* titik pada permukaan dari titik awal. Nilai *geodesic distance* ini disimpan pada variabel T yang dimiliki oleh tiap kelas GeoVertex. Untuk keperluan uji coba, nilai T ini dapat disimpan menjadi sebuah file teks. Selain informasi *geodesic distance*, perangkat lunak ini juga memberikan keluaran berupa titik-titik yang digunakan untuk membentuk *geodesic path*, yang merupakan lintasan pada permukaan yang menghubungkan titik awal dan titik akhir. Informasi daftar titik-titik tersebut disimpan dalam kelas **GeoPoint** yang ada pada kelas **CGeoPath**.

3.2. PERANCANGAN PROSES

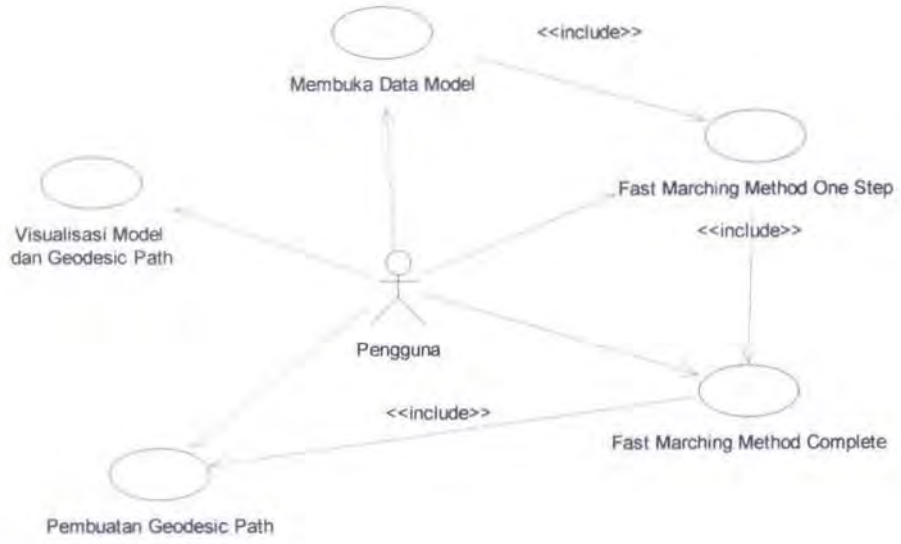


Gambar 3-5 Activity Diagram dari Perangkat Lunak

Secara umum alur aktivitas dari pengguna dalam menjalankan perangkat lunak dapat dilihat pada gambar *activity diagram* di atas. Pada gambar tersebut tampak urutan aktivitas yang dapat dilakukan oleh pengguna melalui perangkat lunak ini. Urutan ini tidak boleh dilanggar. Sebagai contoh jika ingin melakukan proses pembuatan geodesic path maka proses membuka data model sampai dengan proses Fast Marching Method harus telah dilakukan terlebih dahulu.

Dari beberapa aktivitas tersebut dibagi lagi menjadi beberapa proses yang bisa dilakukan oleh pengguna dalam perangkat lunak ini yang diantaranya adalah:

1. Membuka data model.
2. Melakukan proses *Fast Marching Method one step* sebagai langkah perhitungan *geodesic distance*. Proses ini hanya menjalankan hanya sekali tahap perulangan algoritme *Fast Marching Method on Triangulated Domain.*
3. Melakukan proses *Fast Marching Method Complete.* Proses ini menjalankan semua tahap algoritme *Fast Marching Method on Triangulated Domain* sampai selesai.
4. Melakukan proses pembuatan *geodesic path.*
5. Merubah metode visualisasi dari model. Menggambar point, line ,flat polygon maupun smooth polygon (*guround shading*). Selain itu bisa merubah warna dari latar belakang, model, garis dan melakukan transformasi terhadap model meliputi *rotation, translation, spinning,* dan *zooming.*



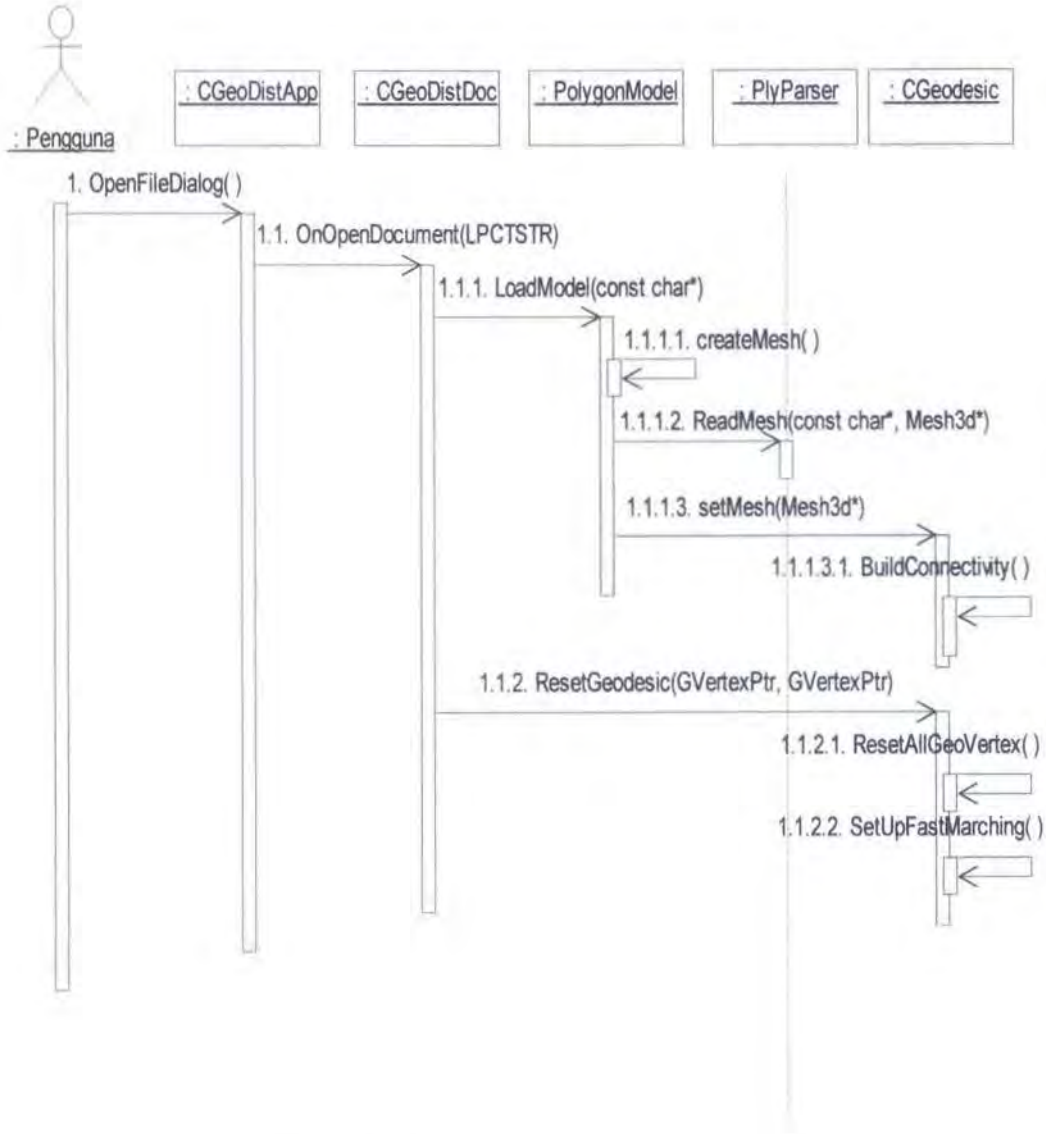
Gambar 3-6 Use Case Diagram proses-proses yang dapat dilakukan oleh pengguna

Gambar 3-6 menunjukkan proses-proses yang dapat dilakukan oleh pengguna. Untuk lebih jelasnya, proses-proses tersebut akan dibahas berikut ini.

3.2.1 Perancangan Proses Buka Data Model

Untuk membuka data model, pengguna meminta kelas `CGeoDistApp` untuk membuka dialog Open File. Setelah menerima nama file yang akan dibuka, kelas `CGeoDistDoc` meminta kelas `PolygonModel` untuk me-load model dengan menjalankan operasi `LoadModel`. Dalam operasi `LoadModel`, `PolygonModel` membuat objek `mesh` baru untuk menampung data model. `PolygonModel` kemudian membuat *parser* (`PlyParser`) untuk dapat membaca data model dengan format ply. Data model kemudian dibaca dengan menjalankan operasi `ReadMesh` pada kelas `PlyParser`. Jika operasi `LoadModel` berhasil dilakukan, `CGeoDistDoc` meminta kelas `CGeodesic` untuk melakukan operasi

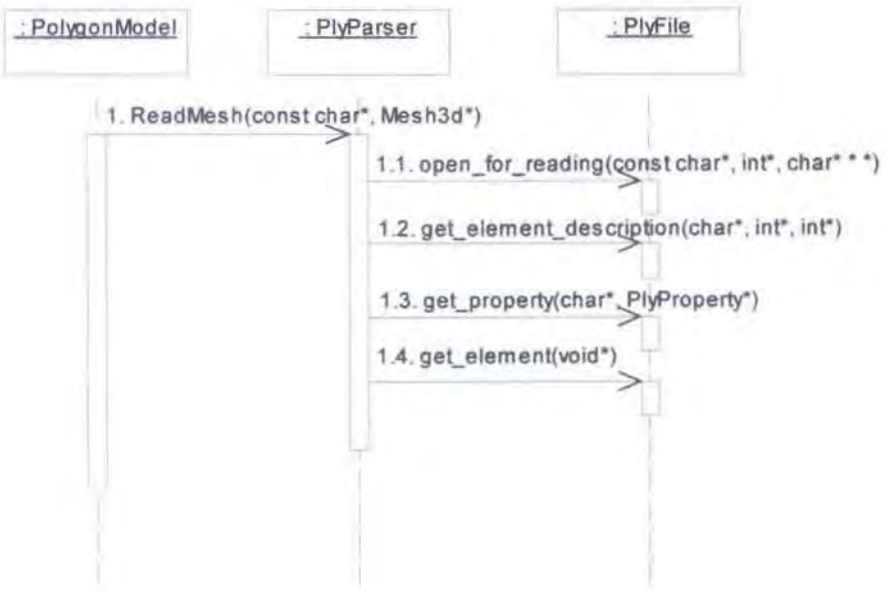
`ResetGeodesic`. Untuk lebih jelasnya, diagram berikut memberikan alur proses pembukaan data model.



Gambar 3-7 Sequence diagram untuk proses pembacaan data model

Seperti dijelaskan pada perancangan data diatas, perangkat lunak ini menggunakan format file ply sebagai format data model. Format file ply merupakan format yang sudah umum digunakan dalam kegiatan penelitian dan

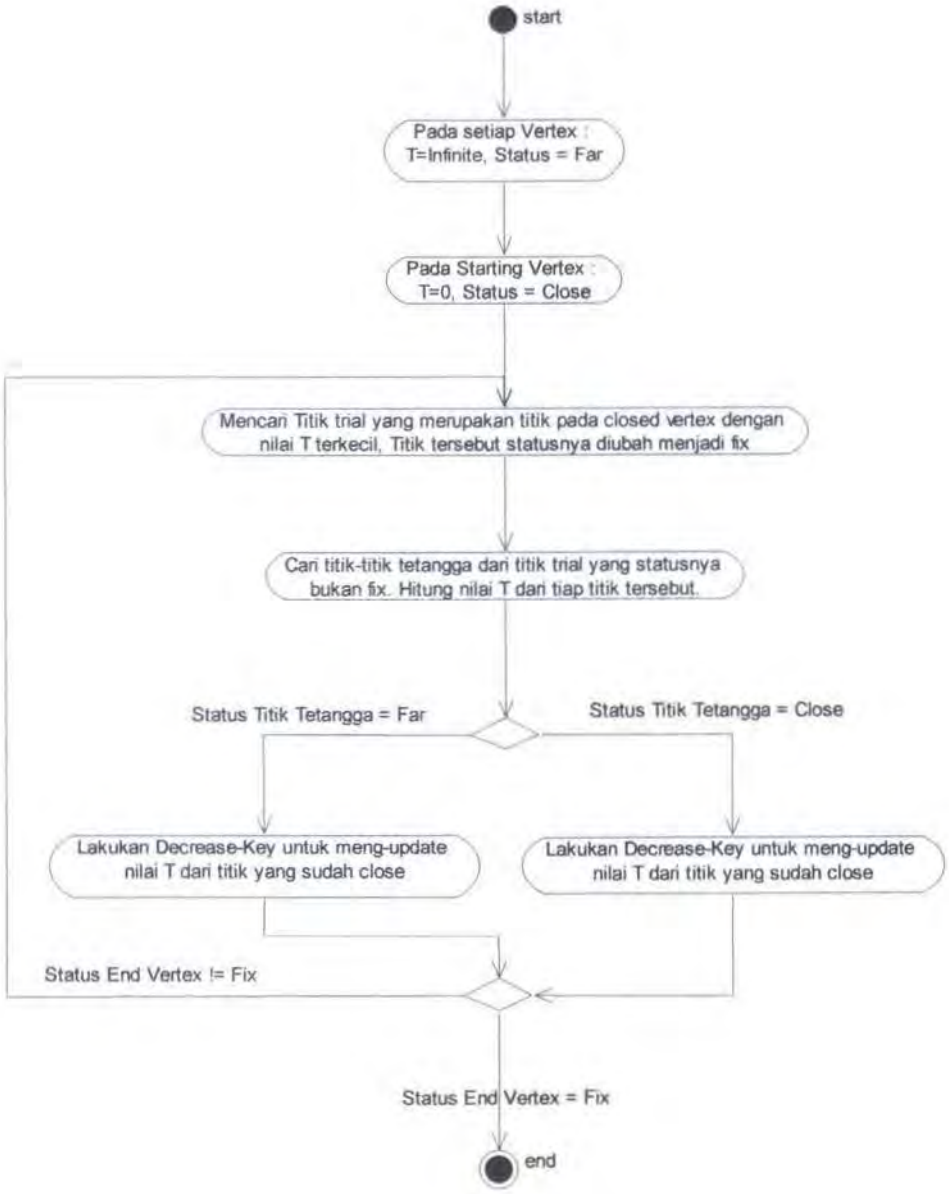
telah tersedia module ply untuk digunakan bagi para peneliti yang di-*release* oleh *Stanford University* dan sifatnya adalah GPL(*General Public Lisence*). Perangkat lunak ini akan menggunakan module tersebut dalam proses pembacaan data model tepatnya pada kelas `PlyParser`. Alur prosesnya digambarkan pada diagram dibawah ini.



Gambar 3-8 Sequence diagram untuk operasi readMesh pada Parser

Operasi `open_for_reading` berfungsi untuk mendapatkan jumlah tipe elemen yang ada di file ply beserta dengan nama elemennya. Untuk setiap elemen yang ada, kemudian dicari jumlah data elemen dan jumlah properti yang dimilikinya dengan menjalankan operasi `get_element_description`. Sebelum data elemen bisa dibaca, dapatkan properti untuk masing-masing elemen dengan menjalankan operasi `get_property`. Setelah semua informasi elemen didapatkan, data elemen bisa dibaca dengan melakukan iterasi sebanyak jumlah data elemen yang bersangkutan dengan menjalankan operasi `get_element`.

3.2.2 Perancangan Proses Perhitungan Geodesic Distance



Gambar 3-9 Activity Diagram dari Proses Perhitungan Geodesic Distance

Seperti yang telah dijelaskan pada bagian sebelumnya, proses perhitungan *geodesic distance* yang dilakukan dalam perangkat lunak ini menggunakan algoritme *Fast Marching Method on Triangulated Domain*. Gambar 3-9 di atas menunjukkan alur secara umum proses perhitungan Geodesic Distance. Dari

urutan alur tersebut dibagi menjadi 2 macam proses dalam menerapkan algoritme *FMM on TD*. Proses-proses tersebut antara lain Proses *Fast Marching Method One Step* yaitu proses *FMM on TD* hanya dengan sekali iterasi dan Proses *Fast Marching Method Complete* yaitu proses *FMM on TD* semua iterasi sampai selesai. Untuk lebih jelasnya proses-proses tersebut akan dijelaskan pada bagian berikut.

3.2.2.1 Perancangan Proses Fast Marching Method One Step

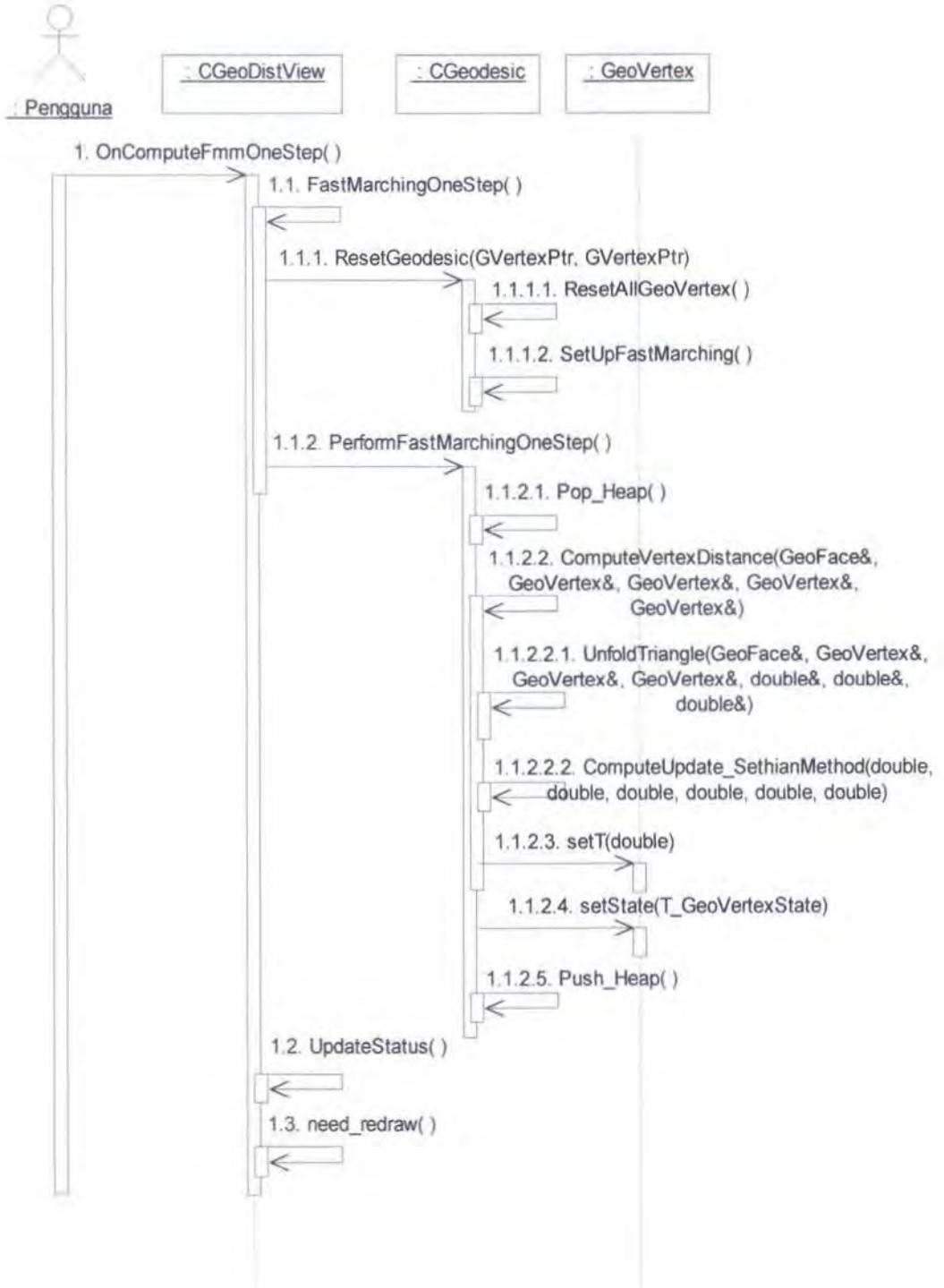
Untuk melakukan proses Fast Marching Method One Step, pengguna meminta kelas `CGeoDistView` untuk melakukan operasi `FastMarchingOneStep`. `CGeoDistView` menjalankan operasi `ResetGeodesic` pada `CGeodesic` apabila proses FMM on TD telah selesai dari proses sebelumnya. Operasi `ResetGeodesic` ini dilakukan untuk melakukan tahap inisialisasi dari proses FMM on TD. Setelah itu dilanjutkan menjalankan operasi `PerformFastMarchingOneStep`.

Di dalam operasi `PerformFastMarchingOneStep`, dijalankan beberapa operasi-operasi penting sesuai dengan sesuai dengan langkah-langkah tahap perulangan dari proses FMM on TD yang telah dijelaskan pada sub bab 2.2.3 Operasi-operasi tersebut antara lain adalah :

- `Pop_Heap` yaitu untuk memilih vertex pada himpunan Closed Vertex yang memiliki nilai T terkecil (sub bab 2.4.2).
- `ComputeVertexDistance` untuk menghitung nilai T dari suatu vertex (sub bab 2.2.3.1).

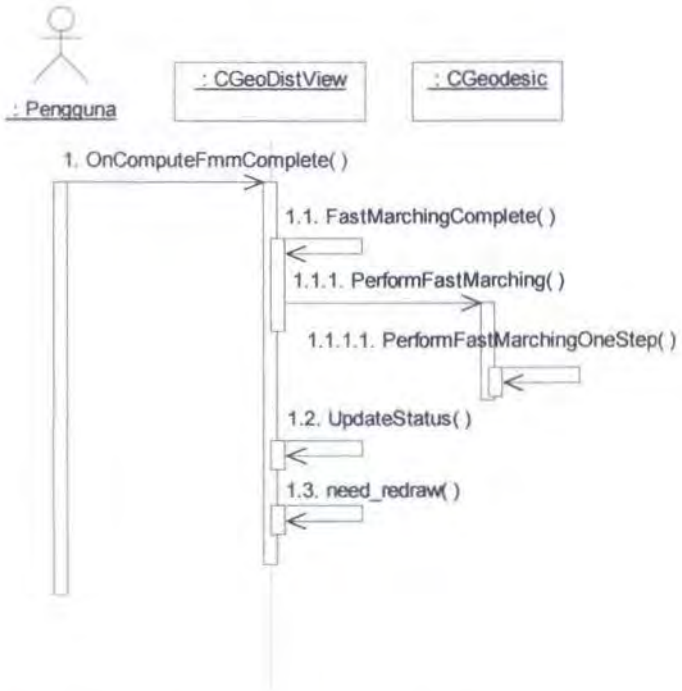
- `UnfoldTriangle` untuk membagi sebuah sudut tumpul menjadi 2 buah sudut lancip. Operasi ini dilakukan jika ditemukan segitiga tumpul pada permukaan (sub bab 2.2.3.3).
- `ComputeUpdate_SethianMethod` untuk menghitung nilai T dari suatu Vertex dengan bantuan nilai T dari 2 vertex lain pada sebuah segitiga (sub bab 2.2.3.2).
- `Push_Heap` untuk memasukkan Closed Vertex yang baru ke dalam himpunan Closed Vertex (sub bab 2.4.2).

Setelah nilai T telah diperoleh maka pada kelas `GeoVertex` dijalankan operasi `setT` untuk menetapkan nilai T dari Vertex tersebut, lalu dijalankan operasi `setState` untuk mengubah status vertex tersebut. Apabila operasi-operasi diatas telah selesai maka dilakukan operasi `update_status` dan `need_redraw` untuk menvisualisasikan hasil proses ini. Untuk lebih jelasnya, alur proses Fast Marching Method One Step bisa dilihat pada gambar 3-10.



Gambar 3-10 Sequence Diagram proses Fast Marching Method One Step

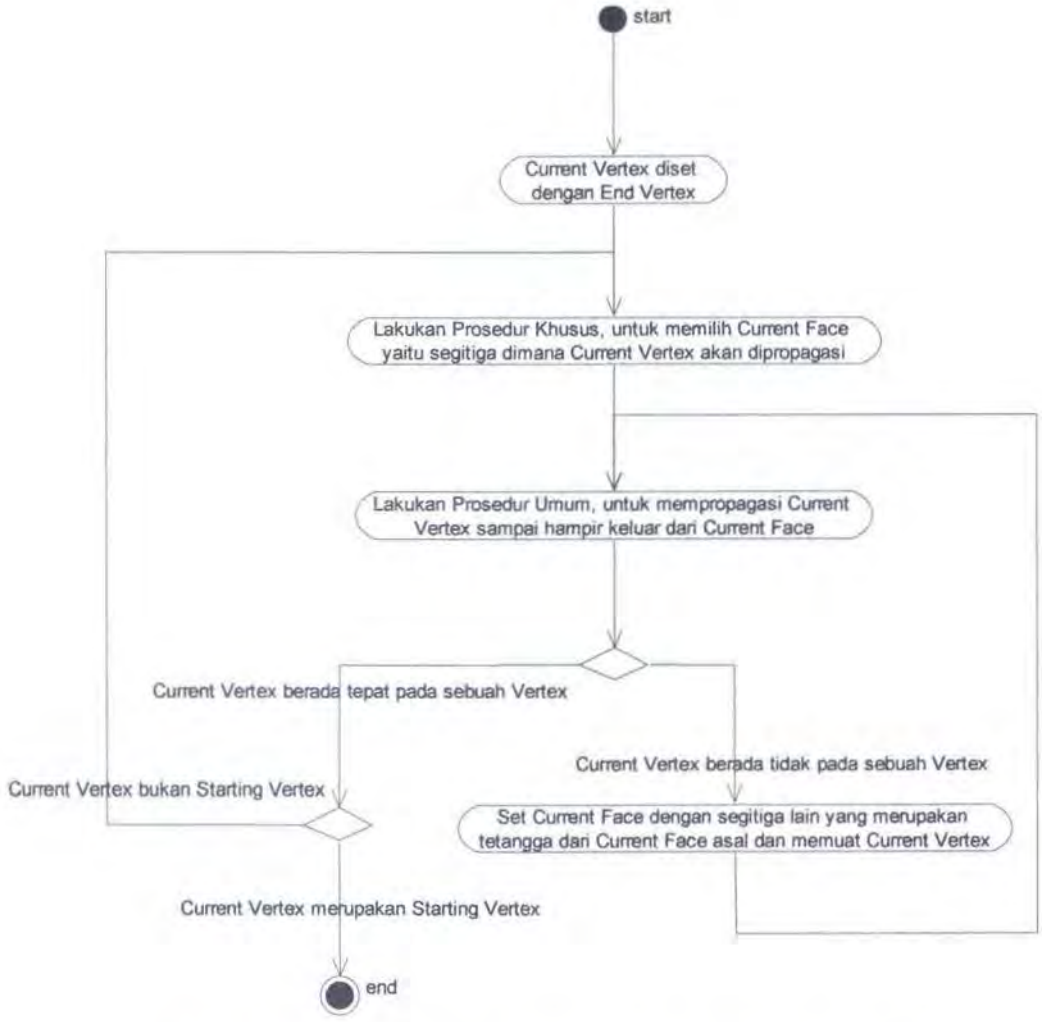
3.2.2.2 Perancangan Proses Fast Marching Method Complete



Gambar 3-11 Sequence Diagram proses Fast Marching Method Complete

Untuk melakukan Proses Fast Marching Method Complete, pengguna meminta kelas `CGeoDistView` untuk melakukan operasi `FastMarchingComplete`. Lalu `CGeoDistView` menjalankan operasi `PerformFastMarching` pada `CGeodesic`. Pada operasi `PerformFastMarching` dilakukan operasi `PerformFastMarchingOneStep` berulang-ulang sampai proses Fast Marching Method telah selesai. Apabila telah selesai maka dilakukan operasi `update_status` dan `need_redraw` untuk menvisualisasikan hasil proses ini. Secara umum, alur proses Fast Marching Method Complete bisa dilihat pada gambar 3-11.

3.2.3 Perancangan Proses Pembuatan Geodesic Path



Gambar 3-12 Activity diagram untuk Proses Pembuatan Geodesic Path

Secara umum alur aktivitas dari proses pembuatan Geodesic Path dapat dilihat pada gambar *activity diagram* di atas. Dari diagram tersebut dibuat alur proses seperti yang tampak pada *sequence diagram* gambar 3-13.

Gambar 3-13 Sequence Diagram proses Pembuatan Geodesic Path

Pada gambar 3-13 tampak bahwa untuk melakukan proses Pembuatan Geodesic Path, pengguna meminta kelas `CGeoDistView` untuk melakukan operasi `ComputeGeodesicPath`. `CGeoDistView` menjalankan operasi `ComputeGeodesicPath` pada `CGeodesic` untuk membentuk Geodesic Path pada kelas `CGeoPath`. Pada operasi `ComputeGeoPath`, kelas `CGeodesic` meminta kelas `CGeoPath` menjalankan operasi `ComputePath`.

Di dalam operasi `ComputePath`, dijalankan beberapa operasi-operasi penting sesuai dengan sesuai dengan langkah-langkah pembuatan geodesic path yang telah dijelaskan pada sub bab 2.3. Operasi-operasi tersebut antara lain adalah :

- `InitPath` yaitu untuk membersihkan kelas `CGeoPath` dan menginisialisasi awal kelas `CGeoPath`, menjadikan titik akhir *(end vertex)* sebagai awal propagasi lintasan.
- `addVertex` yaitu untuk melakukan prosedur khusus (sub bab 2. 3.1).
- `addNewPoint` yaitu untuk melakukan prosedur umum (sub bab 2.3.2).
- `SetUpTriangularInterpolation` yaitu untuk melakukan inisialisasi Triangular Interpolation Quadratic pada segitiga dimana akan dibuat lintasan didalamnya (sub bab 2.3.2.1).
- `ComputeGradient` yaitu untuk mencari nilai $\nabla T(x, y)$ sebagai vektor gradient yang menentukan arah suatu titik yang dipropagasi mundur.

Apabila telah selesai, dilakukan proses `need_redraw` untuk menvisualisasikan Geodesic Path yang telah dibuat.

3.2.4 Proses Visualisasi

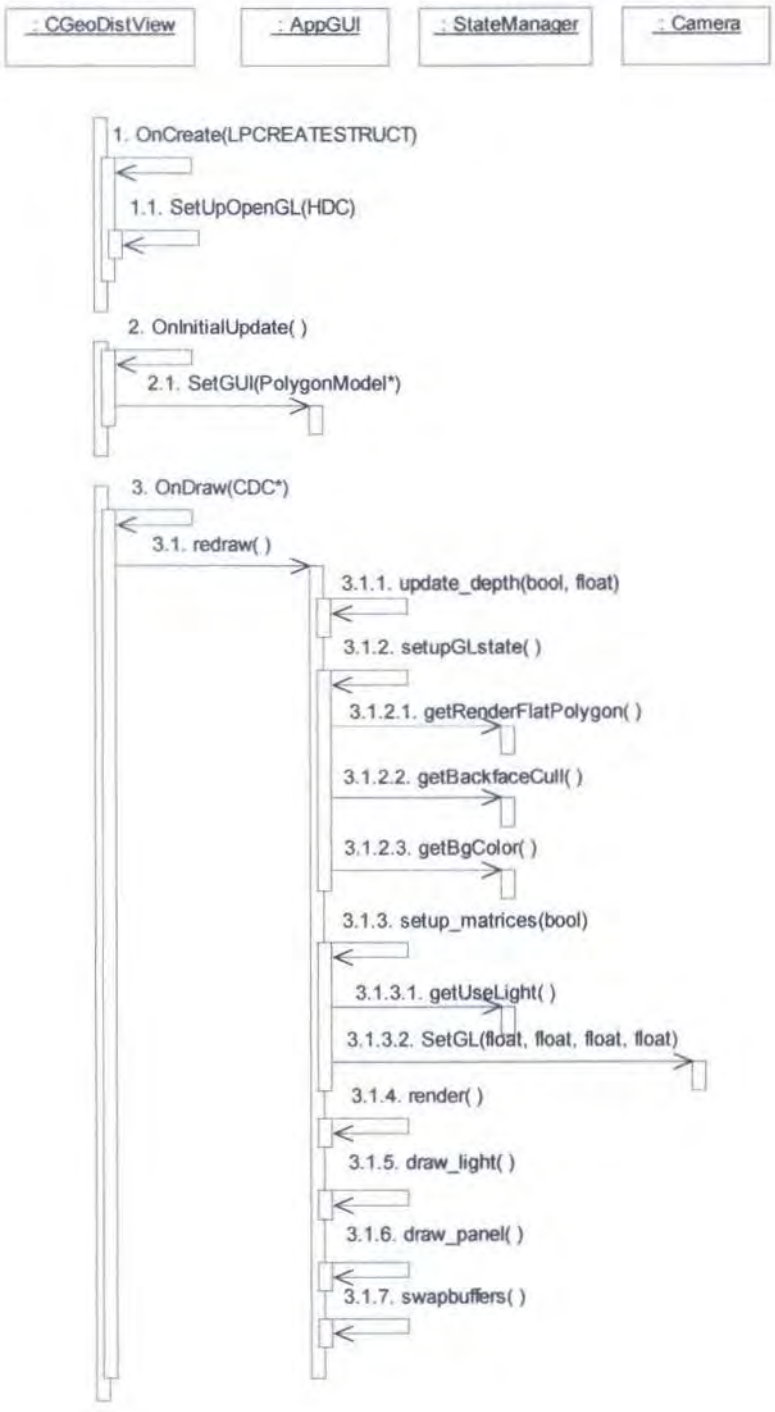
Proses visualisasi terdiri dari empat bagian sub proses yaitu transformasi, pencahayaan, inisialisasi material dan terakhir adalah proses penggambaran pada layar. Agar fungsi OpenGL dapat berjalan dengan baik, device tempat untuk menggambar objek atau dalam hal ini adalah kelas `CGeoDistView` harus diinisialisasi terlebih dahulu. Inisialisasi dilakukan ketika objek view tersebut dibuat dengan operasi `SetUpOpenGL`, kemudian dilakukan inisialisasi pada kelas `AppGUI` melalui operasi `SetGUI`. Proses penggambaran (*rendering*) ke dalam layar dilakukan berulang-ulang setiap kali diminta oleh window atau ketika pengguna merubah arah sudut pandang. Proses tersebut dilakukan pada event `OnDraw`. Proses penggambaran ke dalam buffer dilakukan oleh operasi `redraw` pada kelas `AppGUI`.

Pada operasi `redraw` dilakukan beberapa operasi penting antara lain :

- `update_depth` yaitu untuk meng-update jarak dari pusat camera ke pusat rotasi.
- `setupGLstate` yaitu untuk pengaturan viewport dan properti OpenGL aktivasi.
- `setup_matrices` yaitu untuk pengaturan pencahayaan material dan warna model.
- `SetGL` yaitu untuk pengaturan proyeksi dan rotasi ruang pandang.
- `render` yaitu untuk penggambaran model ke buffer.

- `draw_light` untuk penggambaran cahaya pada buffer.
- `draw_panel` untuk penggambaran panel info pada buffer.
- `swap_buffers` untuk menukar buffer yang aktif dengan buffer yang baru. Dengan adanya fungsi `swap_buffers` ini maka proses visualisasi ini menggunakan teknik *double buffering*. Dengan teknik ini maka efek *flicker* atau efek kedip akan hilang.

Untuk lebih jelasnya, alur proses visualisasi ini dapat dilihat pada gambar 3-14.



Gambar 3-14 Sequence diagram dari Proses Visualisasi

3.3. PERANCANGAN ANTAR MUKA

Gambar 3-15 Rancangan dari drop down menu

Interaksi antara pengguna dengan perangkat lunak menggunakan mouse dan beberapa tombol pada Keyboard. Interaksi ini sebagian besar adalah untuk melakukan proses transformasi dari model seperti rotasi, translasi, zooming maupun proses perubahan posisi sumber cahaya. Untuk melakukan proses buka

data model, proses Fast Marching Method, proses pembuatan Geodesic Path dan perubahan atribut penggambaran dari model menggunakan dropdown Menu seperti yang terlihat pada gambar 3-15.

Untuk melakukan proses membuka dan menutup data model, pengguna dapat memilih menu pada menu dropdown File yaitu Open untuk membuka data model dan Close untuk menutup data model. Perangkat Lunak ini dibuat dengan prinsip MDI *(Multi Document Interface)*, sehingga pengguna dapat membuka beberapa data model sekaligus tanpa harus menutup data model yang lama.

Menu dropdown Render digunakan untuk mengatur visualisasi dari model. Pada group pertama digunakan untuk menentukan tipe penggambaran model dilayar, apakah Point, Line, Flat Polygon, atau Smooth Polygon, dan apakah Vertex Number ditampilkan. Group kedua digunakan untuk menentukan keadaan lingkungan dari penggambaran model, apakah menggunakan cahaya (Use Lighting), kilau (Shiny), putar otomatis (Auto Spin) maupun fasilitas Backface Cull. Menu change color digunakan untuk merubah warna baik warna latar belakang, garis, dan poligon.

Menu dropdown Compute digunakan untuk melakukan proses perhitungan. Pada group pertama digunakan untuk melakukan proses FMM untuk perhitungan Geodesic Distance. Lalu perintah Reset Geodesic untuk melakukan Reset Geodesic dari Model. Pada group berikutnya terdapat perintah-perintah untuk melakukan ujicoba, yaitu menghitung Geodesic Map, menghitung Geodesic Distance dari titik-titik pada permukaan bola (sphere) dengan menggunakan

rumus geometri, menghitung Geodesic Distance dari titik-titik pada permukaan bidang datar (flat surface) dengan menggunakan rumus geometri dan menulis hasil Geodesic Map pada sebuah file teks. Pada group terakhir digunakan untuk melakukan proses pembuatan Geodesic Distance. Pada group ini terdapat perintah Compute Geodesic Path, Show All Geodesic Path, dan Random End Vertex & Compute Geodesic Path.

Menu dropdown view digunakan untuk mengaktifkan atau menon-aktifkan toolbar, status bar, dan console. Sedangkan Menu dropdown window untuk melakukan pengaturan apabila dibuka banyak dokumen (data model).

Menu Command pada menu dropdown Help digunakan untuk menampilkan informasi dalam menggunakan aplikasi ini. Sedangkan menu about untuk menampilkan informasi pembuat aplikasi.

BAB IV
IMPLEMENTASI PERANGKAT LUNAK

Pada bab ini akan diuraikan tentang implementasi dari rancangan perangkat lunak yang telah dibuat pada bab III. Pembahasan meliputi lingkungan pembangunan perangkat lunak, implementasi struktur data, implementasi proses dan implementasi antar muka.

4.1. Lingkungan Pembangunan Perangkat Lunak

Lingkungan pembangunan aplikasi ini meliputi perangkat keras dan perangkat lunak yang digunakan. Spesifikasi perangkat keras dan perangkat lunak yang digunakan dalam pembangunan aplikasi penghitungan *geodesic distance* dan pembuatan *geodesic path* ini dapat dilihat pada tabel dibawah ini.

Tabel 4-1 Lingkungan pembangunan aplikasi

Perangkat Keras	Prosesor	: AMD Duron 1100 MHz
	Memory	: 512 MB
	VGA Card	: GeForce 4 MX 440 64 MB
Perangkat Lunak	Sistem Operasi	: Microsoft Windows 2000 Pro.
	Bahasa Pemrograman	: C++.
	Compiler & Tools	: Microsoft Visual C++ v6.0
	Graphics Library	: Open Graphics Library(OpenGL)

4.2. IMPLEMENTASI STRUKTUR DATA

Pada sub bab ini akan dijelaskan mengenai implementasi dari perancangan data yang telah dibuat pada bab III. Masing-masing kelas akan diperlihatkan atribut yang dimiliki dan fungsi untuk masing-masing atribut.

4.2.1 Kelas Pnt3

```
1  class Pnt3 {
2  public:
3          float v[3];
4
5          //----- member function -----
6          float  dot(const Pnt3 &vec) const;
7          Pnt3   cross(const Pnt3 &vec) const;
8          float  length(void) const;
9          float  square_length(void) const;
10         Pnt3&  normalize(void);
11 };
```

Gambar 4-1 Potongan Kode Deklarasi Kelas Pnt3

Kelas Pnt3 digunakan untuk menyimpan koordinat 3 dimensi dari model . pada kelas ini disediakan fungsi fungsi aritmatika yang digunakan untuk mempermudah proses perhitungan pada vertex. Kordinat x, y, dan z disimpan di array v dengan tipe data float.

4.2.2 Kelas GeoVertex

```
1  class GeoVertex {
2  private:
3          enum T_GeoVertexState{Far,Close,Fix};
4
5          T_GeoVertexState mState;
6          unsigned int mId;
7          Pnt3 mCoord ;
8          Pnt3 mNormal;
9          double mT;
10         vector<GeoFace*> pFaceNeighbors;
11         vector<GeoVertex*>  pVertexNeighbors;
12 };
```

Gambar 4-2 Potongan Kode Deklarasi Kelas GeoVertex

Variabel mState digunakan untuk menyimpan status dari vertex tersebut apakah Far, Close atau Fix. Variabel mId digunakan untuk menyimpan nomor urut vertex dalam vertexlist., mCoord digunakan untuk menyimpan koordinat 3 dimensi dari vertex, mNormal digunakan untuk menyimpan vektor normal dari vertex yang digunakan untuk smooth polygon pada proses rendering model. Variabel mT digunakan untuk menyimpan nilai *geodesic distance* titik tersebut dari titik awal. Variabel pFaceNeighbors merupakan daftar pointer ke objek GeoFace yang menjadi tetangga dari titik ini. Sedangkan variabel pVertexNeighbors merupakan daftar pointer ke objek GeoVertex yang menjadi tetangga dari titik ini.

4.2.3 Kelas GeoFace

```
1  class GeoFace {
2  private:
3          GeoVertex* mpVertex[3];
4          Pnt3       mNormal;
5          GeoFace* mpFaceNeighbors[3];
6          unsigned int mId;
7  static GeoTriangularInterpolation::T_TriangulationInterpolationType
8          TriangulationInterpolationType;
9          GeoTriangularInterpolation* pTriangularInterpolation;
10 };
```

Gambar 4-3 Potongan Kode Deklarasi Kelas GeoFace

Variabel mpVertex digunakan untuk menyimpan vertex yang membentuk objek GeoFace. Data yang disimpan berupa data pointer sebanyak 3 buah dengan tipe GeoVertex. mNormal digunakan untuk menyimpan vektor normal dari face tersebut, vektor normal ini dipakai pada proses render model secara flat polygon. Variabel mpFaceNeighbors digunakan untuk menyimpan face yang menjadi tetangga dari face tersebut. Data yang disimpan berupa data pointer ke objek GeoFace. Variabel mIndex digunakan untuk menyimpan nomor urut face dalam

FaceList. Sedangkan variabel pTriangularInterpolation dan TriangulationInterpolationType untuk menyimpan kelas GeoTriangularInterpolation yaitu kelas yang mengatur interpolasi segitiga dari face tersebut dan tipe interpolasi yang digunakan.

4.2.4 Kelas GeoTriangularInterpolation_Quadratic

```
1  class GeoTriangularInterpolation{
2  public:
3          enum T_TriangulationInterpolationType{kLinear, kQuadratic};
4  };
5
6  class GeoTriangularInterpolation_Quadratic : public
7  GeoTriangularInterpolation
8  {
9  private:
10         double C[6];
11         Pnt3 u,v;
12         Pnt3 w;
13 };
```

Gambar 4-4 Potongan Kode Deklarasi Kelas GeoTriangularInterpolation_Quadratic

Kelas ini digunakan dalam proses pembuatan *Geodesic Path* sebagai data yang menyimpan fungsi *T(x,y)*. Variabel C digunakan untuk menyimpan 6 buah koefisien pembentuk fungsi T. Variabel u, v untuk menyimpan vektor pembentuk basis ortonormal pada face, sedangkan w untuk menyimpan koordinat 3 dimensi dari titik pusat basis ortonormal yang dibuat.

4.2.5 Kelas GeoPoint

```
1  class GeoPoint {
2  private:
3          vector<Pnt3> mSubPointList;
4          GeoVertex* pVert1;
5          GeoVertex* pVert2;
6          double rCoord;
7          GeoFace* pCurFace;
8  };
```

Gambar 4-5 Potongan Kode Deklarasi Kelas GeoPoint

Kelas GeoPoint ini digunakan untuk menyimpan titik awal propagasi yang berada ditengah-tengah edge, dan titik-titik hasil propagasi yang berada di dalam face yang bersangkutan. Variabel mSubPointList merupakan daftar titik pembentuk lintasan hasil propagasi yang berada dalam face tertentu. Variabel pVert1 dan pVert2 merupakan pointer ke objek GeoVertex yang membentuk edge. rCoord merupakan nilai rasio jarak titik tersebut dari vertex pertama yang membentuk edge (pVert1). Sedangkan variabel pCurFace adalah pointer ke objek GeoFace untuk menyimpan Face dimana lintasan ini dibuat.

4.2.6 Kelas CGeoPath

```
1  class CGeoPath  {
2  private:
3          vector<GeoPoint*> mPointList;
4          GeoFace* pCurFace;
5          GeoFace* pPrevFace;
6          double rStepSize;
7  };
```

Gambar 4-6 Potongan Kode Deklarasi Kelas GeoPath

Kelas CGeoPath ini digunakan dalam untuk menyimpan *geodesic path* yang menghubungkan titik awal dan titik akhir. Variabel mPointList digunakan untuk menyimpan data vertex yang membentuk Path dalam bentuk pointer ke objek GeoPoint. Variabel pCurFace dan pPrevFace untuk menyimpan pointer ke Objek GeoFace yang menunjukkan face mana yang sedang dan telah dilewati oleh path. Sedangkan rStepSize adalah konstanta skalar yang menunjukkan step dari penambahan propagasi.

4.2.7 Kelas CGeodesic

```
1  class CGeodesic {
2  private:
3          vector<GeoFace*> mFaceList;
4          vector<GeoVertex*> mVertexList;
```

```
5         vector<GeoVertex*> mCloseVertex;
6
7         GeoVertex* pStart;
8         GeoVertex* pEnd;
9         float mTimeCost;
10        float *mGeoMap;
11
12        bool bIsMarchingEnd;
13        CGeoPath mGeoPath;
14    }
```

Gambar 4-7 Potongan Kode Deklarasi Kelas CGeodesic

Kelas CGeodesic merupakan kelas utama dalam perangkat lunak ini. Kelas ini digunakan untuk menyimpan semua struktur data yang berhubungan dengan proses penghitungan *geodesic distance* dan pembuatan *geodesic path*. Variabel mFaceList merupakan daftar pointer ke objek GeoFace yang menyimpan semua face yang dimiliki oleh model. Variabel mVertexList merupakan daftar pointer ke objek GeoVertex yang menyimpan semua vertex yang dimiliki oleh model. Variabel mCloseVertex merupakan daftar pointer ke objek GeoVertex yang menyimpan Vertex yang statusnya Close. Variabel pStart dan pEnd menyimpan titik awal dan titik akhir. Variable mTimeCost menyimpan waktu yang dibutuhkan untuk melakukan proses. Variabel mGeoMap digunakan untuk menyimpan array yang berisi nilai *geodesic distance* tiap titik dari titik awal. Variabel bIsMarchingEnd adalah variabel bertipe boolean untuk menunjukkan apakah proses FMM sudah selesai atau belum. Sedangkan mGeoPath digunakan untuk menyimpan data *geodesic path* yang dibuat.

4.3. IMPLEMENTASI PROSES

Pada sub bab ini akan dijelaskan mengenai implementasi dari perancangan proses yang telah dibuat pada bab III. Implementasi ini meliputi implementasi

proses buka data model, implementasi proses perhitungan geodesic distance, implementasi proses pembuatan geodesic path, dan implementasi proses visualisasi model ke layar.

4.3.1 Proses Buka Data Model

Untuk proses buka data model, terlebih dahulu dibutuhkan nama file ply yang menyimpan data model tersebut. Berikut adalah implementasi dialog untuk memasukkan nama file :

```
1  CString CGeoDistApp::OpenFileDialog()
2  {
3     static char szFilter[256];
4     strcpy(szFilter,"Ply  Files (*.ply)|*.ply|");
5     CFileDialog filedlg(TRUE,NULL,NULL,OFN_FILEMUSTEXIST |
6     FN_PATHMUSTEXIST | OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
7        szFilter,this->m_pActiveWnd);
8
9     filedlg.m_ofn.lpstrTitle = "Open";
10    if(filedlg.DoModal()==IDOK){
11       return filedlg.GetFileName();
12    }
13    else
14       return "";
15 }
```

Gambar 4-8 Potongan Kode Fungsi OpenFileDialog pada Kelas CGeoDistApp

Setelah mendapatkan nama file yang akan dibuka, maka akan dibuat objek model, kemudian data model ini di-load dengan menggunakan fungsi **`LoadModel`** yang dimiliki oleh objek model. Berikut adalah fungsi untuk load model.

```
1  bool PolygonModel::LoadModel(const char *_pmodelfilename){
2     PlyParser    *pParser;
3     Mesh3dPtr    pNewMesh;
4
5     bool       retval;
6
7     //----- buat mesh3d baru -----
8     pNewMesh = createMesh();
9     //----- buat parser -----
10    pParser = new PlyParser;
11    retval = pParser->ReadMesh(_pmodelfilename,pNewMesh);
12    if(retval && setMesh(pNewMesh)){
13       //----- ambil nama model
14       char* resVal;
```

```
15        if (resVal    = (strrchr(_pmodelfilename,'\\'))){
16           _pmodelfilename = resVal+1;
17        }
18        strcpy(modelName, _pmodelfilename);
19        return true;
20     }
21     else{
22        return false;
23     }
24  };
```

Gambar 4-9 Potongan Kode Fungsi LoadModel pada Kelas PolygonModel

Pada saat me-*load* model, data model akan dibaca dengan menjalankan fungsi **ReadMesh**. Objek ini memanfaatkan library ply yang ada. Berikut adalah fungsi untuk membaca data model dari file ply.

```
 1  bool PlyParser::ReadMesh(const char *_infile, Mesh3d *_pmesh){
 2     int i,j;
 3     int nelems;     //jml element dalam file
 4     char **elist;
 5     PlyFile ply;
 6
 7     if (ply.open_for_reading((char*)_infile, &nelems, &elist) == 0)
 8        return 0;
 9
10     set_offsets();
11     char *elem_name;   //nama elemen
12     int num_elems;  //jml item untuk setiap element
13     int nprops;     //jumlah property
14
15     for(i = 0;i < nelems; i++) {
16        elem_name = elist[i];
17        ply.get_element_description(elem_name,    &num_elems,    &nprops);
18
19        if(equal_strings("vertex",elem_name)){
             if(ply.is_valid_property("vertex",vert_prop_x.name) &&
                ply.is_valid_property("vertex",vert_prop_y.name) &&
                ply.is_valid_property("vertex",vert_prop_z.name)) {
20              ply.get_property(elem_name,&vert_prop_x);
21              ply.get_property(elem_name,&vert_prop_y);
22              ply.get_property(elem_name,&vert_prop_z);
23           }
24
25           PlyVertex pVert;
26           for(j = 0;j < num_elems; j++){
27              ply.get_element((void*) &pVert);
28              VERTEX3D *vert;
29              vert = _pmesh->makeVertex(pVert.coord);
30           }
31        }
32
33        if(equal_strings("face",elem_name)){
34           if(ply.is_valid_property("face",face_props[0].name)){
35              ply.get_property("face",&face_props[0]);
36           }
37
```

```
38          if(num_elems == 0)
39            continue;
40
41          PlyFace pFace;
42          for(j = 0;j < num_elems; j++){
43            ply.get_element_noalloc((void *) &pFace);
44            if(pFace.nverts != 3)
45              continue;
46
47            FACE3D *tri;
48            tri = _pmesh->makeFace(_pmesh->getVertex(pFace.verts[0]),
                                     _pmesh->getVertex(pFace.verts[1]),
                                     _pmesh->getVertex(pFace.verts[2]));
49          }
50        }
51      }
52      _pmesh->calculateNormalPerVertex();
53      return 1;
54  };
```

Gambar 4-10 Potongan Kode Fungsi ReadMesh pada Kelas PlyParser

Setelah data model berhasil dibaca, kemudian objek **CGeodesic** di-set dengan mesh hasil pembacaan. Proses ini dilakukan oleh fungsi **setMesh** pada kelas **CGeodesic**.

```
1  void CGeodesic::setMesh(Mesh3d *mesh)
2  {
3    GVertexPtr pv;
4
5    FOR_EACH_VERTEX3D(mesh->getVertexList(),itv)
6      pv = makeVertex(P(itv)->Coord());
7      addVertex(pv);
8    END
9
10   GFacePtr pf;
11   GVertexPtr v1,v2,v3;
12   unsigned int i=0;
13
14   FOR_EACH_FACE3D(mesh->getFaceList(),itf)
15     v1 = getVertex(P(itf)->v1()->Index());
16     v2 = getVertex(P(itf)->v2()->Index());
17     v3 = getVertex(P(itf)->v3()->Index());
18     pf = makeFace(v1,v2,v3);
19     addFace(pf);
20     if(v1->getFace()==NULL)
21       v1->setFace(mFaceList[i]);
22     if(v2->getFace()==NULL)
23       v2->setFace(mFaceList[i]);
24     if(v3->getFace()==NULL)
25       v3->setFace(mFaceList[i]);
26     i++;
27   END
28   BuildConnectivity();
29 }
```

Gambar 4-11 Potongan Kode Fungsi setMesh pada Kelas CGeodesic

Di dalam fungsi **setMesh** dijalankan fungsi **BuildConnectivity** yaitu untuk membangun vertex dan face tetangga. Berikut adalah fungsi **BuildConnectivity**.

```
1  void CGeodesic::BuildConnectivity()
2  {
3     FOR_EACH_LFACE(mFaceList,itf)
4        GFacePtr pFace = (GFacePtr) *itf;
5        for(unsigned long i=0;i<3;i++)
6        mVertexList[pFace->v(i)->Index()]->
                                    pFaceNeighbors.push_back(pFace);
7     END
8
9     GVertexPtr va,vb;
10    FOR_EACH_LFACE(mFaceList,itf3)
11       for(int iEdge=0; iEdge<3; iEdge++){
12          va = (*itf3)->v(iEdge+1);
13          vb = (*itf3)->v(iEdge+2);
14          (*itf3)->v(iEdge)->addVertexNeighbor(va);
15          (*itf3)->v(iEdge)->addVertexNeighbor(vb);
16       }
17    END
18
19    FOR_EACH_LFACE(mFaceList,itf2)
20       GFacePtr pFace = (GFacePtr) *itf2;
21
22       vector<GFacePtr>* pFaceList[3];
23
24       for(unsigned long i=0;i<3;i++){
25          GVertexPtr pVert = pFace->v(i);
26          pFaceList[i] = &(mVertexList[pVert->Index()]->
                                                pFaceNeighbors);
27       }
28
29       for(i=0;i<3;i++)
30       {
31          GFacePtr pNeighbor = NULL;
32          unsigned long i1=(i+1)%3;
33          unsigned long i2=(i+2)%3;
34
35          bool bFind = false;
36
37          for(LFITERATOR it1=pFaceList[i1]->begin();
                 it1!=pFaceList[i1]->end() && bFind!=true;++it1){
38             GFacePtr pFace1 = *it1;
39             for(LFITERATOR it2=pFaceList[i2]->begin();
                    it2!=pFaceList[i2]->end() && bFind!=true;++it2){
40                GFacePtr pFace2 = *it2;
41                if( pFace1==pFace2 && pFace1!=pFace){
42                   pNeighbor= pFace1;
43                   bFind=true;
44                }
45             }
46          }
47
48          pFace->setFaceNeighbor(pNeighbor,i);
49
50          if(pNeighbor){
```

```
51              long nEdgeNumber = pNeighbor->getEdgeNumber(*pFace->v(i1),
                                                            *pFace->v(i2));
52              ASSERT(nEdgeNumber>=0);
53              pNeighbor->setFaceNeighbor(pFace,nEdgeNumber);
54          )
55      }
56    END
57  }
```

Gambar 4-12 Potongan Kode Fungsi BuildConnectivity pada Kelas CGeodesic

4.3.2 Proses Perhitungan Geodesic Distance

Proses perhitungan Geodesic Distance dilakukan dengan menggunakan algoritme Fast Marching Method yang mempunyai 2 tahap yaitu tahap inisialisasi dan tahap perulangan. Untuk melakukan tahap inisialisasi dijalankan fungsi **`ResetGeodesic`**. Berikut adalah fungsi untuk melakukan tahap inisialisasi FMM on TD.

```
1  void CGeodesic::ResetGeodesic(GVertexPtr _start,GVertexPtr _end)
2  {
3    //tiap vertex T=inf, status=far
4    FOR_EACH_LVERTEX(mVertexList,itv)
5      GVertexPtr pVert = (GeoVertex *) *itv;
6      pVert->ResetGeoVertex();
7    END
8
9    setStart(_start);
10   setEnd(_end);
11
12   SetUpFastMarching();
13 }
```

Gambar 4-13 Potongan Kode Fungsi ResetGeodesic pada Kelas CGeodesic

Untuk mengimplementasikan tahap perulangan FMM on TD, digunakan fungsi **`PerformFastMarchingMethodOneStep`**. Berikut adalah fungsi untuk melakukan tahap perulangan FMM on TD.

```
1  bool CGeodesic::PerformFastMarchingOneStep(bool bAll)
2  {
3    if(mCloseVertex.empty()){
4      this->bIsMarchingEnd=true;
5      return true;   //finish
6    }
7    ASSERT(bIsMarchingBegin);
8
```

```
 9    GVertexPtr pCurVert = mCloseVertex.front();
10    ASSERT(pCurVert !=NULL);
11
12    std::pop_heap(mCloseVertex.begin(),mCloseVertex.end(),
                                                  CompareVertex);
13
14    mCloseVertex.pop_back();
15
16    pCurVert->setState(GeoVertex::Fix);
17
18    if(pCurVert == pEnd && !bAll){
19       return true;    //finish
20    }
21
22    FOR_EACH_LVERTEX(pCurVert->pVertexNeighbors,itv)
23       GVertexPtr pNewVert = *itv;//pFace->v(temp+i);
24       ASSERT(pNewVert!=NULL);
25
26       double rNewDistance = GEO_INFINITE;
27
28       FOR_EACH_LFACE(pNewVert->pFaceNeighbors,itf)
29          GFacePtr pFace = (GFacePtr) *itf;
30          ASSERT(pFace!=NULL);
31
32          GVertexPtr pVert1 = (GVertexPtr) pFace->
                                        getNextVertex(*pNewVert);
33          ASSERT(pVert1!=NULL);
34
35          GVertexPtr pVert2 = (GVertexPtr) pFace->
                                        getNextVertex(*pVert1);
36          ASSERT(pVert2!=NULL);
37
38          if(pVert1->T() > pVert2->T()){
39             GVertexPtr pTempVert = pVert1;
40             pVert1=pVert2;
41             pVert2=pTempVert;
42          }
43          //Menghitung Nilai T
44          rNewDistance = MIN(rNewDistance,ComputeVertexDistance(*pFace,
                  *pNewVert,*pVert1,*pVert2,*pCurVert->getFront()));
45       END

46       switch(pNewVert->getState()){
47          case GeoVertex::Close:
48             if(rNewDistance<=pNewVert->T())
49             {
50               pNewVert->setT(rNewDistance);
51               pNewVert->setFront(pCurVert->getFront());
52               std::make_heap(mCloseVertex.begin(),mCloseVertex.end(),
                                                  CompareVertex);
53             }
54          break;
55          case GeoVertex::Far:
56             pNewVert->setT(rNewDistance);
57             mCloseVertex.push_back(pNewVert);
58             std::push_heap(mCloseVertex.begin(),mCloseVertex.end(),
                                                  CompareVertex);
59             pNewVert->setState(GeoVertex::Close);
60             pNewVert->setFront(pCurVert->getFront());
61          break;
62          case GeoVertex::Fix:
63
```

```
64          break;
65          default:
66            ASSERT(false);
67        }
68    END
69    return bIsMarchingEnd;
70  }
```

Gambar 4-14 Potongan Kode Fungsi PerformFastMarchingOneStep pada Kelas CGeodesic

Di dalam fungsi **PerformFastMarchingOneStep** dijalankan fungsi **ComputeVertexDistance** yaitu fungsi untuk menghitung nilai T atau nilai geodesic distance dari suatu titik terhadap titik awal.

```
1  double CGeodesic::ComputeVertexDistance(GeoFace &CurrentFace,
                      GeoVertex &CurrentVertex, GeoVertex &Vert1,
                      GeoVertex &Vert2, GeoVertex &CurrentFront)
2  {
3     int F = 1;
4
5     if(Vert1.getState()!=GeoVertex::Far ||
                                  Vert2.getState() != GeoVertex::Far)
6     {
7        Pnt3 Edge1 = Vert1.Coord() - CurrentVertex.Coord();
8        double b = Edge1.length();
9        Edge1 /= b;
10       Pnt3 Edge2 = Vert2.Coord() - CurrentVertex.Coord();
11       double a = Edge2.length();
12       Edge2 /= a;
13       double d1 = Vert1.T();
14       double d2 = Vert2.T();
15
16       bool bVert1Usable = Vert1.getState() != GeoVertex::Far &&
                                  Vert1.getFront() == &CurrentFront;
17       bool bVert2Usable = Vert2.getState() != GeoVertex::Far &&
                                  Vert2.getFront() == &CurrentFront;
18
19       if(!bVert1Usable && bVert2Usable){
20          return d2 + a * F;
21       }
22       if(bVert1Usable && !bVert2Usable){
23          return d1 + b * F;
24       }
25       if(bVert1Usable && bVert2Usable){
26          double dot= Edge1.dot(Edge2);
27
28          if(dot<0 && bUseUnfolding )
29            double c,dot1,dot2;
30            GVertexPtr pVert = UnfoldTriangle(CurrentFace,
                            CurrentVertex,Vert1,Vert2,c,dot1,dot2);
31            if(pVert!=NULL && pVert->getState()!=GeoVertex::Far)
32            {
33              double d3 = pVert->T();
34              double t;
35              t=ComputeUpdate_SethianMethod(d1,d3,c,b,dot1,F);
36              t=MIN(t,ComputeUpdate_SethianMethod(d3,d2,a,c,dot2,F));
37              return t;
```

```
38              }
39          }
40          return ComputeUpdate_SethianMethod(d1,d2,a,b,dot,F);
41      }
42  }
43  return GEO_INFINITE;
44  }
```

Gambar 4-15 Potongan Kode Fungsi ComputeVertexDistance pada Kelas CGeodesic

Di dalam fungsi `ComputeVertexDistance` dijalankan fungsi `UnfoldTriangle` untuk membagi sebuah sudut tumpul menjadi 2 buah sudut lancip. Fungsi ini dilakukan jika ditemukan segitiga tumpul pada permukaan.

```
1  GVertexPtr CGeodesic::UnfoldTriangle(GeoFace &CurFace,
                  GeoVertex &vert,GeoVertex &vert1, GeoVertex &vert2,
                              double &dist, double &dot1, double &dot2)
2  {
3     Pnt3& v = vert.Coord();
4     Pnt3& v1 = vert1.Coord();
5     Pnt3& v2 = vert2.Coord();
6
7     Pnt3& e1 = v1-v;
8     double rNorm1 = e1.length();
9     e1 /= rNorm1;
10
11    Pnt3& e2 = v2-v;
12    double rNorm2 = e2.length();
13    e2 /= rNorm2;
14
15    double dot = e1.dot(e2);
16
17    Pnt2 eq1 = Pnt2(dot,sqrtf(1-dot*dot));
18    Pnt2 eq2 = Pnt2(1,0);
19
20    Pnt2 x1(rNorm1,0);
21    Pnt2 x2 = eq1 * rNorm2;
22
23    Pnt2 xstart1 = x1;
24    Pnt2 xstart2 = x2;
25
26    GVertexPtr pV1 = &vert1;
27    GVertexPtr pV2 = &vert2;
28
29    GFacePtr pCurFace = (GFacePtr) CurFace.getFaceNeighbor(vert);
30
31    unsigned int nNum=0;
32    while(nNum<50 && pCurFace!=NULL){
33       GVertexPtr pV = (GVertexPtr) pCurFace->getVertex(*pV1,*pV2);
34       ASSERT(pV!=NULL);
35
36       e1 = pV2->Coord() - pV1->Coord();
37       double rNorm1 = e1.length();
38       e1 /= rNorm1;
39
40       e2 = pV->Coord() - pV1->Coord();
41       double rNorm2 = e2.length();
```

```
42        e2 /= rNorm2;
43
44        Pnt2 vv = (x2-x1)*rNorm2/rNorm1;
45        dot = e1.dot(e2);
46        Pnt2 x = vv.Rotate(-acos(dot)) + x1;
47
48        double lambda11 = - (x1.dot(eq1)) / ( (x-x1).dot(eq1) );
49        double lambda12 = - (x1.dot(eq2)) / ( (x-x1).dot(eq2) );
50        double lambda21 = - (x2.dot(eq1)) / ( (x-x2).dot(eq1) );
51        double lambda22 = - (x2.dot(eq2)) / ( (x-x2).dot(eq2) );
52
53        bool bIntersect11 = (lambda11>=0) && (lambda11<=1);
54        bool bIntersect12 = (lambda12>=0) && (lambda12<=1);
55        bool bIntersect21 = (lambda21>=0) && (lambda21<=1);
56        bool bIntersect22 = (lambda22>=0) && (lambda22<=1);
57
58        if(bIntersect11 && bIntersect12){
59           pCurFace = (GFacePtr) pCurFace->getFaceNeighbor(*pV2);
60           pV2 = pV;
61           x2=x;
62        }else if(bIntersect21 && bIntersect22){
63           pCurFace = (GFacePtr) pCurFace->getFaceNeighbor(*pV1);
64           pV1 = pV;
65           x1 = x;
66        }else {
67           dist = x.length();
68           dot1 = x.dot(xstart1) / (dist * xstart1.length());
69           dot2 = x.dot(xstart2) / (dist * xstart2.length());
70           return pV;
71        }
72        nNum++;
73     }
74     return NULL;
75  }
```

Gambar 4-16 Potongan Kode Fungsi UnfoldTriangle pada Kelas CGeodesic

Selain itu di dalam fungsi **`ComputeVertexDistance`** juga dijalankan fungsi **`ComputeUpdate_SethianMethod`** untuk menghitung nilai T dari suatu Vertex dengan bantuan nilai T dari 2 vertex lain pada sebuah segitiga sesuai dengan rumus yang telah dijelaskan pada sub bab 2.2.3.2.

```
1  double CGeodesic::ComputeUpdate_SethianMethod(double d1, double d2,
                              double a, double b, double dot, double F)
2  {
3     double t = GEO_INFINITE;
4     double rCosAngle = dot;
5     double rSinAngle2 = 1-dot*dot;
6     double u = d2-d1;
7
8     double f2 = a*a + b*b - 2*a*b*rCosAngle;
9     double f1 = b * u * (a*rCosAngle-b);
10    double f0 = b * b * (u*u - F*F*a*a*rSinAngle2);
11    double delta = f1*f1 - f0*f2;
12
```

```
13     if(delta >=0) {
14        if(MYABS(f2)>GEO_EPSILON){
15           t = (-f1 - sqrtf(delta))/f2;
16
17           if(t<u || b*(t-u)/t<a*rCosAngle || a/rCosAngle<b*(t-u)/t){
18              t= (-f1 + sqrtf(delta))/f2;
19           }
20        }
21        else {
22           if(f1!=0)
23              t = -f0/f1;
24           else
25              t = -GEO_INFINITE;
26        }
27     }
28     else
29        t=-GEO_INFINITE;
30
31     if(u<t && a*rCosAngle < b*(t-u)/t && b*(t-u)/t < a/rCosAngle ){
32        return t+d1;
33     }else {
34        return MIN(b*F+d1,a*F+d2);
35     }
36  }
```

Gambar 4-17 Potongan Kode Fungsi ComputeUpdate_SethianMethod pada Kelas CGeodesic

Fungsi **`PerformFastMarchingOneStep`** hanya mengimplementasikan tahap perulangan FMM on TD sebanyak sekali perulangan. Untuk mengimplementasikan tahap perulangan FMM on TD sampai selesai atau sampai nilai T pada titik akhir (end vertex) diperoleh digunakan fungsi **`PerformFastMarching`**. Fungsi ini sebenarnya hanyalah menjalankan fungsi **`PerformFastMarchingOneStep`** secara berulang-ulang sampai selesai.

```
1  void CGeodesic::PerformFastMarching(bool bAll)
2  {
3     int begin, ends;
4     begin = clock ();
5     while( !this->PerformFastMarchingOneStep(bAll) ){ }
6     ends = clock ();
7     mTimeCost = ( float ) ( ends - begin );
8  }
```

Gambar 4-18 Potongan Kode Fungsi PerformFastMarching pada Kelas CGeodesic

4.3.3 Proses Pembuatan Geodesic Path

Untuk melakukan proses pembuatan Geodesic Path pengguna meminta kelas `CGeoDistView` untuk menjalankan fungsi `ComputeGeodesicPath`. Pada fungsi tersebut kelas `CGeoDistView` menjalankan fungsi `ComputePath` pada kelas `CGeoPath`. Inti dari proses pembuatan Geodesic Path dilakukan oleh fungsi `ComputePath` tersebut.

```
1  void CGeoPath::ComputePath(GeoVertex &StartVert,
                                        unsigned int nMaxLength)
2  {
3     this->InitPath(StartVert);
4     unsigned int nNum=0;
5     while(this->addNewPoint()==0 && nNum<nMaxLength) {
6        nNum++;
7     }
8  }
```

Gambar 4-19 Potongan Kode Fungsi ComputePath pada Kelas CGeoPath

Pada fungsi `ComputePath` ini dijalankan fungsi `InitPath` yaitu fungsi untuk menginisialisasi awal kelas `CGeoPath`. Inisialisasi yang dilakukan adalah membersihkan kelas `CGeoPath` dan menempatkan titik akhir (*end vertex*) sebagai titik awal propagasi.

```
1  void CGeoPath::InitPath(GeoVertex &StartVert)
2  {
3     this->removePointList();
4     this->addVertex(StartVert);
5  }
```

Gambar 4-20 Potongan Kode Fungsi InitPath pada Kelas CGeoPath

Fungsi `removePointList` digunakan untuk membersihkan kelas `CGeoPath` dari hasil proses pembuatan Geodesic Path sebelumnya.

```
1  void CGeoPath::removePointList()
2  {
3     GPointIt itp;
4     FOR_EACH(itp, mPointList)
5        if((*itp)) {
6           (*itp)->mSubPointList.clear();
```

```
7           delete (*itp);
8         }
9       END
10      mPointList.clear();
11  }
```

Gambar 4-21 Potongan Kode Fungsi removePointList pada Kelas CGeoPath

Sedangkan Fungsi **addVertex** digunakan untuk mengimplementasi prosedur khusus seperti yang telah dijelaskan di subbab 2.3.1.

```
1   void CGeoPath::addVertex(GeoVertex &Vert)
2   {
3     pPrevFace = pCurFace;
4     double rBestT = GEO_INFINITE;
5     pCurFace = NULL;
6     GVertexPtr pSelectedVert = NULL;
7
8     for(GeoVertexIterator it = Vert.BeginVertexIterator();
                                    it!=Vert.EndVertexIterator();++it)
9     {
10      GVertexPtr pVert = (GVertexPtr) *it;
11
12      if(pVert->T() < rBestT){
13        rBestT = pVert->T();
14        pSelectedVert = pVert;
15        GVertexPtr pVert1 = (GVertexPtr) it.GetLeftVertex();
16        GVertexPtr pVert2 = (GVertexPtr) it.GetRightVertex();
17        if(pVert1!=NULL && pVert2!=NULL){
18          if(pVert1->T() < pVert2->T())
19            pCurFace = (GFacePtr) it.GetLeftFace();
20          else
21            pCurFace = (GFacePtr) it.GetRightFace();
22        }else if(pVert1!=NULL){
23          pCurFace = (GFacePtr) it.GetLeftFace();
24        }else{
25          pCurFace = (GFacePtr) it.GetRightFace();
26        }
27      }
28    }
29    GeoPoint* pPoint = new GeoPoint;
30    mPointList.push_back(pPoint);
31    pPoint->setVertex1(Vert);
32    pPoint->setVertex2(*pSelectedVert);
33    pPoint->setCoord(1);
34    pPoint->setCurFace(*pCurFace);
35  }
```

Gambar 4-22 Potongan Kode Fungsi addVertex pada Kelas CGeoPath

Setelah fungsi **InitPath** selesai dilakukan maka dijalankan fungsi **addNewPoint** sebagai implementasi dari prosedur umum yang telah dijelaskan pada sub bab 2.3.2.

```
1  int CGeoPath::addNewPoint()
2  {
3     GPointPtr pPoint = mPointList.back();
4
5     GVertexPtr pVert1 = pPoint->getVertex1();
6     GVertexPtr pVert2 = pPoint->getVertex2();
7     GVertexPtr pVert3=(GVertexPtr)pCurFace->
                                        getVertex(*pVert1,*pVert2);
8
9     T_SubPointList& SubPointVector = pPoint->getSubPointList();
10    double x,y,z;
11
12    x = pPoint->getCoord();
13    y = 1-x;
14    z = 0;
15
16    double l1 = (pVert1->Coord() - pVert3->Coord()).length();
17    double l2 = (pVert2->Coord() - pVert3->Coord()).length();
18
19    pCurFace->SetUpTriangularInterpolation();
20
21    unsigned int nNum=0;
22    while(nNum<1000){
23       nNum++;
24       double dx,dy;
25       double l,a;
26
27       pCurFace->ComputeGradient(*pVert1,*pVert2,*pVert3,x,y,dx,dy);
28
29       if(MYABS(dx)>GEO_EPSILON){
30          l = l1 * x/dx;
31          a = y-l*dy/l2;
32          if(l>0 && l<=rStepSize && 0<=a && a<=1){
33             GPointPtr pNewPoint = new GeoPoint;
34             mPointList.push_back(pNewPoint);
35             pNewPoint->setVertex1(*pVert2);
36             pNewPoint->setVertex2(*pVert3);
37             pNewPoint->setCoord(a);
38             pPrevFace = pCurFace;
39
40             if(pCurFace->getFaceNeighbor(*pVert2,*pVert3)==NULL){
41                pNewPoint->setCurFace(*pCurFace);
42                return 0;
43             }
44             pCurFace= (GFacePtr)pCurFace->
                                  getFaceNeighbor(*pVert2,*pVert3);
45             pNewPoint->setCurFace(*pCurFace);
46
47             if(a<0.01 && pVert2->T()<GEO_EPSILON)
48                return -1;
49             if(a>0.99 && pVert3->T()<GEO_EPSILON)
50                return -1;
51             return 0;
52          }
53       }
54
55       if(MYABS(dy)>GEO_EPSILON){
56          l=l2*y/dy;
57          a = x-l*dx/l1;
58          if( l>0 && l<=rStepSize && 0<=a && a<=1){
59             GPointPtr pNewPoint = new GeoPoint;
60             mPointList.push_back(pNewPoint);
```

```
61            pNewPoint->setVertex1(*pVert1);
62            pNewPoint->setVertex2(*pVert3);
63            pNewPoint->setCoord(a);
64            pPrevFace = pCurFace;
65
66            if(pCurFace->getFaceNeighbor(*pVert1,*pVert3)==NULL){
67               pNewPoint->setCurFace(*pCurFace);
68               return 0;
69            }
70            pCurFace=(GFacePtr)pCurFace->
                                    getFaceNeighbor(*pVert1,*pVert3);
71            pNewPoint->setCurFace(*pCurFace);
72
73            if(a<0.01 && pVert1->T()<GEO_EPSILON)
74               return -1;
75            if(a>0.99 && pVert3->T()<GEO_EPSILON)
76               return -1;
77            return 0;
78         }
79      }
80
81      if(MYABS(dx/l1+dy/l2)>GEO_EPSILON) {
82         l = -z/(dx/l1+dy/l2);
83         a = x-l*dx/l1;
84         if( l>0 && l<=rStepSize && 0<=a && a<=1 ){
85            GPointPtr pNewPoint = new GeoPoint;
86            mPointList.push_back(pNewPoint);
87            pNewPoint->setVertex1(*pVert1);
88            pNewPoint->setVertex2(*pVert2);
89            pNewPoint->setCoord(a);
90            pPrevFace = pCurFace;
91
92            if(pCurFace->getFaceNeighbor(*pVert1,*pVert2)==NULL){
93               pNewPoint->setCurFace(*pCurFace);
94               return 0;
95            }
96            pCurFace=(GFacePtr) pCurFace->
                              getFaceNeighbor(*pVert1,*pVert2);
97            pNewPoint->setCurFace(*pCurFace);
98
99            if(a<0.01 && pVert1->T()<GEO_EPSILON)
100              return -1;
101           if(a>0.99 && pVert2->T()<GEO_EPSILON)
102              return -1;
103           return 0;
104        }
105     }
106
107     if(MYABS(dx)<GEO_EPSILON && MYABS(dy)<GEO_EPSILON){
108        GVertexPtr pSelectedVert = pVert1;
109        if(pVert2->T() < pSelectedVert->T())
110           pSelectedVert = pVert2;
111        if(pVert3->T() < pSelectedVert->T())
112           pSelectedVert = pVert3;
113
114        this->addVertex(*pSelectedVert);
115        if(pSelectedVert->T()<GEO_EPSILON)
116           return -1;
117        if(pCurFace==pPrevFace && pPoint->getCoord()>1-GEO_EPSILON){
118           return -1;
119        }
120        return 0;
```

```
121         }
122
123         double xprev = x, yprev = y;
124
125         x = x - rStepSize*dx/l1;
126         y = y - rStepSize*dy/l2;
127
128         if( x<0 || x>1 || y<0 || y>1)
129         {
130           GFacePtr pNextFace=(GFacePtr) pCurFace->
                                          getFaceNeighbor(*pVert3);
131
132           if(pNextFace==pPrevFace ||
                     pCurFace->getFaceNeighbor(*pVert3)==NULL)
133           {
134             GVertexPtr pSelectedVert = pVert1;
135             if(pVert2->T() < pSelectedVert->T())
136               pSelectedVert=pVert2;
137             if(pVert3->T() < pSelectedVert->T())
138               pSelectedVert=pVert3;
139
140             this->addVertex(*pSelectedVert);
141
142             if(pSelectedVert->T() < GEO_EPSILON)
143               return -1;
144             if(pCurFace==pPrevFace&&pPoint->getCoord()>1-GEO_EPSILON){
145               return -1;
146             }
147           }else{
148             pPrevFace = pCurFace;
149             pCurFace = (GFacePtr) pCurFace->getFaceNeighbor(*pVert3);
150             GPointPtr pNewPoint = new GeoPoint;
151             mPointList.push_back(pNewPoint);
152             pNewPoint->setVertex1(*pVert1);
153             pNewPoint->setVertex2(*pVert2);
154             pNewPoint->setCurFace(*pCurFace);
155             pNewPoint->setCoord(xprev);
156           }
157           return 0;
158         }
159         Pnt3 temp(x,y,1-x-y);
160         SubPointVector.push_back(temp);
161       }
162       GVertexPtr pSelectedVert = pVert1;
163
164       if(pVert2->T()<pSelectedVert->T())
165         pSelectedVert = pVert2;
166       if(pVert3->T()<pSelectedVert->T())
167         pSelectedVert = pVert3;
168
169       this->addVertex(*pSelectedVert);
170
171       if(pSelectedVert->T() < GEO_EPSILON)
172         return -1;
173       if(pCurFace == pPrevFace && pPoint->getCoord() > 1-GEO_EPSILON)
174         return-1;
175       return 0;
176   }
```

Gambar 4-23 Potongan Kode Fungsi addNewPoint pada Kelas CGeoPath

Pada fungsi **addNewPoint** dijalankan fungsi **SetUpTriangular Interpolation** sebagai implementasi proses inisialisasi Triangular Interpolation Quadratic pada segitiga dimana akan dibuat lintasan didalamnya seperti yang dijelaskan pada sub bab 2.3.2.1.

```
1  void GeoTriangularInterpolation_Quadratic::
                        SetUpTriangularInterpolation(GeoFace& Face)
2  {
3     GVertexPtr pV0 = (GVertexPtr) Face.getVertex(0);
4     GVertexPtr pV1 = (GVertexPtr) Face.getVertex(1);
5     GVertexPtr pV2 = (GVertexPtr) Face.getVertex(2);
6
7     GFacePtr pFace0 = Face.getFaceNeighbor(0);
8     GFacePtr pFace1 = Face.getFaceNeighbor(1);
9     GFacePtr pFace2 = Face.getFaceNeighbor(2);
10
11    GVertexPtr pW0=NULL;
12    GVertexPtr pW1=NULL;
13    GVertexPtr pW2=NULL;
14
15    if(pFace0!=NULL)
16       pW0=(GVertexPtr) pFace0->getVertex(*pV1,*pV2);
17    if(pFace1!=NULL)
18       pW1=(GVertexPtr) pFace1->getVertex(*pV0,*pV2);
19    if(pFace2!=NULL)
20       pW2=(GVertexPtr) pFace2->getVertex(*pV0,*pV1);
21
22    Pnt3 V0 = pV0->Coord();
23    Pnt3 V1 = pV1->Coord();
24    Pnt3 V2 = pV2->Coord();
25
26    Pnt3 W0,W1,W2;
27
28    if(pW0!=NULL)
29       W0 = pW0->Coord();
30    else
31       W0 = (V1+V2)*0.5;
32
33    if(pW1!=NULL)
34       W1 = pW1->Coord();
35    else
36       W1 = (V0+V2)*0.5;
37
38    if(pW2!=NULL)
39       W2 = pW2->Coord();
40    else
41       W2 = (V0+V1)*0.5;
42
43    Pnt3 e0 = V0-V2;
44    Pnt3 e1 = V1-V2;
45    Pnt3 e2 = V1-V0;
46
47    Pnt3 s0 = W0 - V2;
48    Pnt3 s1 = W1 - V2;
49    Pnt3 s2 = W2 - V0;
50
```

```
51      double l0 = e0.length();
52      double l1 = e1.length();
53      double l2 = e2.length();
54      double m0 = s0.length();
55      double m1 = s1.length();
56      double m2 = s2.length();
57
58      u = e0/l0;
59      v = ( (u.cross(e1)).cross(u));
60      v.normalize();
61      w = V2;
62
63      double a = acos( e0.dot(e1)/(l0*l1));
64      double b = acos( e1.dot(s0)/(l1*m0));
65      double c = acos( e0.dot(s1)/(l0*m1));
66      double d = acos(-e0.dot(e2)/(l0*l2));
67      double e = acos( e2.dot(s2)/(l2*m2));
68
69      double Points[6][2];
70
71      Points[0][0] = l0;
72      Points[0][1] = 0;
73
74      Points[1][0] = l1*cos(a);
75      Points[1][1] = l1*sin(a);
76
77      Points[2][0] = 0;
78      Points[2][1] = 0;
79
80      Points[3][0] =  m0*cos(a+b);
81      Points[3][1] =  m0*sin(a+b);
82
83      Points[4][0] =  m1*cos(c);
84      Points[4][1] = -m1*sin(c);
85
86      Points[5][0] =  l0 - m2*cos(d+e);
87      Points[5][1] =  m2*sin(d+e);
88
89      double Values[6];
90      Values[0] = pV0->T();
91      Values[1] = pV1->T();
92      Values[2] = pV2->T();
93
94      if( pW0!=NULL )
95         Values[3] = pW0->T();
96      else
97         Values[3] = (Values[1]+Values[2])*0.5;
98
99      if( pW1!=NULL )
100        Values[4] = pW1->T();
101     else
102        Values[4] = (Values[0]+Values[2])*0.5;
103
104     if( pW2!=NULL )
105        Values[5] = pW2->T();
106     else
107        Values[5] = (Values[1]+Values[0])*0.5;
108
109     Fit2ndOrderPolynomial2D( Points, Values, C);
110  }
```

Gambar 4-24 Potongan Kode Fungsi SetUpTriangularInterpolation

Setelah fungsi `SetUpTriangularInterpolation` berhasil dilakukan maka pada fungsi `addNewPoint` dilanjutkan dengan melakukan fungsi `ComputeGradient` untuk menghitung nilai $\nabla T(x, y)$ pada suatu titik di dalam suatu segitiga. Dengan bantuan nilai $\nabla T(x, y)$ maka proses propagasi titik dapat dilakukan sampai titik tersebut keluar dari face.

```
1  void GeoTriangularInterpolation_Quadratic::ComputeGradient(
     GeoVertex& v0 ,GeoVertex& v1,GeoVertex& v2, double x, double y,
     double &dx,double &dy)
2  {
3     Pnt3& e = v2.Coord();
4     Pnt3 e0 = v0.Coord() - e;
5     Pnt3 e1 = v1.Coord() - e;
6
7     Pnt3 trans = e-w;
8
9     double p00 = e0.dot(u);
10    double p01 = e1.dot(u);
11    double p10 = e0.dot(v);
12    double p11 = e1.dot(v);
13
14    double s = x*p00 + y*p01 + trans.dot(u);
15    double t = x*p10 + y*p11 + trans.dot(v);
16
17    double gu=C[1] + C[3]*t + C[4]*2*s;
18    double gv=C[2] + C[3]*s + C[5]*2*t;
19
20    double rDet = p00*p11 - p01*p10;
21    ASSERT(rDet!=0);
22    if(MYABS(rDet)>GEO_EPSILON){
23       dx = 1/rDet * ( p11*gu - p01*gv) * e0.length();
24       dy = 1/rDet * (-p10*gu + p00*gv) * e1.length();
25    }
26    else
27       dx=dy=0;
28 }
```

Gambar 4-25 Potongan Kode Fungsi ComputeGradient

4.3.4 Proses Visualisasi

Dalam pemrograman grafik dengan menggunakan library OpenGL, sebelum fungsi-fungsinya dapat digunakan, maka device tempat untuk menggambar objek harus diinisialisasi terlebih dahulu. Terdapat perbedaan antara GDI device sebagai

device default dalam pemrograman Win32 yang biasa disebut *device context*, dalam OpenGL objek digambar di device yang disebut *rendering context*.

Sebuah rendering context dibuat dengan menggunakan fungsi `wglCreateContext`. Dalam perangkat lunak ini, rendering device dibuat didalam fungsi **`SetUpOpenGL`**.

```
1  BOOL CGeoDistView::SetUpOpenGL(HDC hdc)
2  {
3     PIXELFORMATDESCRIPTOR *ppfd;
4     int pixelformat;
5
6     PIXELFORMATDESCRIPTOR pfd = { sizeof(PIXELFORMATDESCRIPTOR), 1,
         PFD_DRAW_TO_WINDOW | PFD_SUPPORT_OPENGL | PFD_DOUBLEBUFFER,
         PFD_TYPE_RGBA, 24 , 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         24, 8, 0, PFD_MAIN_PLANE, 0, 0, 0,0
      };
7
8     ppfd = &pfd;
9
10    if ( (pixelformat = ChoosePixelFormat(hdc, ppfd)) == 0 ) {
11       ::MessageBox(NULL, "ChoosePixelFormat failed", "Error", MB_OK);
12       return FALSE;
13    }
14
15    if (SetPixelFormat(hdc, pixelformat, ppfd) == FALSE) {
16       ::MessageBox(NULL, "SetPixelFormat failed", "Error", MB_OK);
17       return FALSE;
18    }
19
20    return TRUE;
21 }
```

Gambar 4-26 Potongan Kode Fungsi SetUpOpenGL pada kelas CGeoDistView

Dengan memanggil fungsi **`SetUpOpenGL`** pada saat window dibuat, kemudian dengan menjalankan perintah `wglMakeCurrent` untuk mengaktifkan *Rendering Context*, maka fungsi-fungsi OpenGL siap untuk digunakan.

```
1  int CGeoDistView::OnCreate(LPCREATESTRUCT lpCreateStruct)
2  {
3     if (CView::OnCreate(lpCreateStruct) == -1)
4        return -1;
5
6     HWND hWnd = GetSafeHwnd();
7     m_hgldc = ::GetDC(hWnd);
8     if(!SetUpOpenGL(m_hgldc)) {
9        ::MessageBox(::GetFocus(),"SetUpOpenGL Failed!","Error",MB_OK);
10       return -1;
```

```
11      }
12
13      m_hglrc = wglCreateContext(m_hgldc);
14      int i= wglMakeCurrent(m_hgldc,m_hglrc);
15      SetTimer(1000,30,NULL);
16
17      return 0;
18  }
```

Gambar 4-27 Potongan Kode Fungsi OnCreate pada kelas CGeoDistView

Untuk menampilkan model di layar, aplikasi memanggil fungsi **need_redraw** secara berulang-ulang setiap kali dibutuhkan. Baik ketika event **OnDraw** maupun setiap kali pengguna merubah arah sudut pandang.

```
1  void CGeoDistView::need_redraw()
2  {
3      HWND hWnd = GetSafeHwnd();
4      HDC hDC = ::GetDC(hWnd);
5
6      wglMakeCurrent(hDC,m_hglrc);
7      redraw();
8  }
```

Gambar 4-28 Potongan Kode Fungsi need_redraw pada kelas CGeoDistView

Proses penggambaran ke dalam buffer dan pengaturan terhadap properti OpenGL dilakukan oleh fungsi **redraw** pada **AppGui**.

```
1  void AppGUI::redraw()
2  {
3      // validasi model
4      if (!theModel) {
5          glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
6          swapbuffers();
7          return;
8      }
9
10     if (first_time) {
11         resetviewer();
12         first_time = false;
13     }
14
15     update_depth();
16
17     // Set viewport dan clear screen
18     setupGLstate();
19     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
20
21     // Set up projection, modelview matrices, dan lighting
22     setup_matrices(theState->getUseLight());
23     mFrustum.CalculateFrustum();
24
25     render();
```

```
26      draw_light();
27
28      if(!theModelStatus->bIsSelecting && theState->getShowPanel())
29         draw_panel();
30
31      swapbuffers();
32      return;
33   }
```

Gambar 4-29 Potongan Kode Fungsi redraw pada kelas AppGUI

Pada fungsi **redraw** dijalankan fungsi **update_depth** yaitu untuk meng-update jarak dari pusat camera ke pusat rotasi.

```
 1  void AppGUI::update_depth(bool update_rot_depth, float hint)
 2  {
 3     if (!theModel)
 4        return;
 5
 6     setup_matrices(false);
 7     float depth=0;
 8     if (hint + surface_depth > 1.0e-5 * theModel->getRadius())
 9        depth = surface_depth + hint;
10
11     if (!depth)
12        return;
13
14     surface_depth = depth;
15     if (!update_rot_depth)
16        return;
17
18     // Rotasi disekitar permukaan jika kamera dekat ke permukaan,
19     // jika tidak rotasi terhadap pusat model
20     float dist2center = Dist(thecamera.pos,theModel->getCenter());
21     rot_depth=min(depth*(1.0f+DEPTH_FUDGE*thecamera.fov),
                                                  dist2center);
22  }
```

Gambar 4-30 Potongan Kode Fungsi update_depth pada kelas AppGUI

Setelah fungsi **update_depth** selesai dijalankan, fungsi redraw menjalankan fungsi **setupGLstate**. Pada fungsi **setupGLstate** dilakukan pengaturan viewport dari layar. Selain itu dilakukan pengaturan terhadap beberapa properti *rendering* openGL seperti properti **DEPTH_TEST**. Pengaturan shading model dilakukan apakah model digambar dengan menggunakan *flat shading* atau *smooth shading*. Penentuan apakah model digambar dengan menggunakan backface cull atau tidak serta penentuan warna background juga diatur pada fungsi ini.

```
1  void AppGUI::setupGLstate()
2  {
3     glViewport(viewportx(), viewporty(), screenx(), screeny());
4
5     glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
6     glDepthMask(GL_TRUE);
7     glEnable(GL_DEPTH_TEST);
8     glDepthFunc(GL_LESS);
9     glDisable(GL_DITHER);
10    glDisable(GL_BLEND);
11    glDisable(GL_LIGHTING);
12
13    if(theState->getRenderSmoothPolygon())
14       glShadeModel(GL_SMOOTH);
15    else
16       glShadeModel(GL_FLAT);
17
18    if (theState->getBackfaceCull()) {
19       glEnable(GL_CULL_FACE);
20    } else {
21       glDisable(GL_CULL_FACE);
22    }
23    glClearColor(theState->getBgColor(0),theState->getBgColor(1),
         theState->getBgColor(2),theState->getBgColor(3));
24
25    glClearDepth(1.0);
26
27    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
28 }
```

Gambar 4-31 Potongan Kode Fungsi setupGLstate pada kelas AppGUI

Setelah fungsi **setupGLstate** selesai dijalankan, fungsi **redraw** menjalankan fungsi **setup_matrices** yaitu untuk pengaturan pencahayaan material dan warna model.

```
1  void AppGUI::setup_matrices(bool dolighting)
2  {
3     // Reset Model Matrix
4     glMatrixMode(GL_PROJECTION);
5     glLoadIdentity();
6     glMatrixMode(GL_MODELVIEW);
7     glLoadIdentity();
8
9     // Set up posisi cahaya sebelum loading modelview matrix
10    if (dolighting) {
11       GLfloat light0_position[] = { lightdir[0], lightdir[1],
                                       lightdir[2], 0 };
12       glLightfv(GL_LIGHT0, GL_POSITION, light0_position);
13    }
14
15    // Set up projection n modelview matrices
16    thecamera.SetGL(surface_depth/DOF, surface_depth*DOF,
                                       screenx(), screeny());
17    // Set up pencahayaan
18    if (dolighting) {
19       static float nocolor[4] =  { 0, 0, 0, 1 };
```

```
20        GLfloat mat_shininess[]      = { theState->getShininess() };
21        GLfloat global_ambient[] = { AMBIENT, AMBIENT, AMBIENT, 1 };
22        GLfloat *light0_ambient  = theState->getLightAmbient();
23        GLfloat *light0_diffuse  = theState->getLightDiffuse();
24        GLfloat *light0_specular = theState->getLightSpecular();
25        GLfloat *diffuse             = theState->getDiffuse();
26        GLfloat *mat_specular = theState->getSpecular();
27
28        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE,
                                                        diffuse);
29        glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, mat_specular);
30        glMaterialfv(GL_BACK, GL_SPECULAR, nocolor);
31        glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, mat_shininess);
32        glLightfv(GL_LIGHT0, GL_AMBIENT, light0_ambient);
33        glLightfv(GL_LIGHT0, GL_DIFFUSE, light0_diffuse);
34        glLightfv(GL_LIGHT0, GL_SPECULAR, light0_specular);
35        glLightModelfv(GL_LIGHT_MODEL_AMBIENT, global_ambient);
36        glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
37        glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
38        glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
39        glEnable(GL_COLOR_MATERIAL);
40        glEnable(GL_LIGHTING);
41        glEnable(GL_LIGHT0);
42    }
43 }
```

Gambar 4-32 Potongan Kode Fungsi setup_matrices pada kelas AppGUI

Pengaturan proyeksi layar dilakukan dengan menggunakan objek **camera** yang dimiliki oleh aplikasi yaitu dengan memanggil fungsi **setGL** pada objek **camera**.

```
1  void Camera::SetGL(float neardist, float fardist, float width,
                                                        float height)
2  {
3     float diag = sqrtf(sqr(width) + sqr(height));
4     float top = height/diag * 0.5f*fov * neardist;
5     float bottom = -top;
6     float right = width/diag * 0.5f*fov * neardist;
7     float left = -right;
8
9     glMatrixMode(GL_PROJECTION);
10    glFrustum(left, right, bottom, top, neardist, fardist);
11
12    glMatrixMode(GL_MODELVIEW);
13    glRotatef(180.0f/M_PI*rot, rotaxis[0], rotaxis[1], rotaxis[2]);
14    glTranslatef(-pos[0], -pos[1], -pos[2]);
15 }
```

Gambar 4-33 Potongan Kode Fungsi SetGL pada kelas Camera

Setelah berhasil menjalankan fungsi **update_depth**, **setupGLstate**, dan **setup_matrices**. Maka fungsi **redraw** menjalankan fungsi **render** untuk melakukan proses penggambaran model pada buffer.

```
1  void CGeoDistView::render()
2  {
3     CGeoDistDoc* pDoc = GetDocument();
4
5     bool fix=false;
6
7     Mesh3d *_pmesh = (Mesh3d *)pDoc->pModel->mesh;
8
9     //draw faces
10    if(theState->getRenderFlatPolygon() ||
                  theState->getRenderSmoothPolygon() )
11    {
12       glPolygonMode(GL_FRONT_AND_BACK,GL_FILL);
13       glEnable(GL_POLYGON_OFFSET_FILL);
14       glPolygonOffset(1.0,1.0);
15
16       FOR_EACH_FACE3D(_pmesh->getFaceList(),itf)
17          fix=false;
18          GFacePtr pFace = (GFacePtr) pDoc->pGeo->getFace(
                                                (*itf)->Index());
19          for(int i=0;i<3;i++){
20             if(pFace->getVertex(i)->getState()==GeoVertex::Fix)
21                fix=true;
22          }
23          if(fix)
24             glColor3f(0,1,1);
25          else
26             glColor4fv(theState->getModelColor());
27
28          glBegin(GL_TRIANGLES);
29          if(theState->getRenderFlatPolygon())
30             glNormal3fv((*itf)->Normal().v );
31
32          if(theState->getRenderSmoothPolygon())
33             glNormal3fv((*itf)->mpVertex[0]->mNormal.v);
34          glVertex3fv((*itf)->v1()->Coord().v);
35
36          if(theState->getRenderSmoothPolygon())
37             glNormal3fv((*itf)->mpVertex[1]->mNormal.v);
38          glVertex3fv((*itf)->v2()->Coord().v);
39
40          if(theState->getRenderSmoothPolygon())
41             glNormal3fv((*itf)->mpVertex[2]->mNormal.v);
42          glVertex3fv((*itf)->v3()->Coord().v);
43          glEnd();
44       END
45    }
46    //draw line
47    if(theState->getRenderLine())
48    {
49       glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
50       FOR_EACH_FACE3D(_pmesh->getFaceList(),itf2)
51          glLineWidth(1);
52          glColor4fv(theState->getLineColor());
```

```
53          glBegin(GL_TRIANGLES);
54          glNormal3fv((*itf2)->Normal().v );
55          glVertex3fv((*itf2)->v1()->Coord().v);
56          glVertex3fv((*itf2)->v2()->Coord().v);
57          glVertex3fv((*itf2)->v3()->Coord().v);
58          glEnd();
59        END
60      }
61
62      //draw  vertex
63      if(theState->getRenderPoint())
64      {
65        FOR_EACH_LVERTEX(pDoc->pGeo->mVertexList,itv2)
66          GVertexPtr pVert = (GeoVertex*) *itv2;
67          glPointSize(10);
68          glDisable( GL_LIGHTING );
69          glBegin( GL_POINTS );
70
71          switch(pVert->getState()){
72            case GeoVertex::Fix:  //fix
73              glColor3f(1,1,0);
74            break;
75            case GeoVertex::Far: // far
76              glColor3f(0,1,0);
77            break;
78            case GeoVertex::Close: //close
79              glColor3f(1,0,0);
80            break;
81          }
82          glNormal3fv(pVert->mNormal.v);
83          glVertex3fv(pVert->Coord().v);
84          glEnd();
85
86          glEnable( GL_LIGHTING );
87        END
88        //draw trial point
89        GVertexPtr trial=pDoc->pGeo->getTrialVertex();
90        if(trial){
91          glDisable( GL_LIGHTING );
92          glPointSize(12);
93          glColor3f(1,1,0);
94          glBegin( GL_POINTS );
95            glVertex3d(trial->Coord().v[0],trial->Coord().v[1],
                                              trial->Coord().v[2] );
96          glEnd();
97          glEnable( GL_LIGHTING );
98        }
99
100       if(!theState->getRenderFlatPolygon() &&
                              !theState->getRenderSmoothPolygon() )
101       {
102         glDisable( GL_LIGHTING );
103         glPolygonMode(GL_FRONT_AND_BACK,GL_FILL);
104         FOR_EACH_FACE3D(_pmesh->getFaceList(),itf0)
105           glBegin(GL_TRIANGLES);
106           glColor4fv(theState->getBgColor());
107
108           glNormal3fv((*itf0)->Normal().v );
109           glVertex3fv((*itf0)->v1()->Coord().v);
110           glVertex3fv((*itf0)->v2()->Coord().v);
111           glVertex3fv((*itf0)->v3()->Coord().v);
112           glEnd();
```

```
113         END
114         glEnable( GL_LIGHTING );
115       }
116     }
117     //draw vertex number
118     if(theState->getRenderVertexNumber())
119     {
120       FOR_EACH_LVERTEX(pDoc->pGeo->mVertexList,itv3)
121         GVertexPtr pVert = (GeoVertex*) *itv3;
122         gprintf(m_hgldc,pVert->Coord().v[0],pVert->Coord().v[1],
                              pVert->Coord().v[2],"%d",pVert->Index());
123       END
124     }
125
126     //draw starting point
127     GVertexPtr start=pDoc->pGeo->getStart();
128
129     glDisable( GL_LIGHTING );
130     glPointSize(15);
131     glColor3f(0,0,1);
132     glBegin( GL_POINTS );
133       glVertex3d(start->Coord().v[0],start->Coord().v[1],
                                              start->Coord().v[2] );
134     glEnd();
135     glEnable( GL_LIGHTING );
136
137     //draw end point
138     GVertexPtr end=pDoc->pGeo->getEnd();
139     glDisable( GL_LIGHTING );
140     glPointSize(15);
141     glColor3f(1,0,1);
142     glBegin( GL_POINTS );
143     glVertex3d(end->Coord().v[0],end->Coord().v[1],
                                              end->Coord().v[2] );
144     glEnd();
145     glEnable( GL_LIGHTING );
146
147     //draw Geo Path;
148     if(pDoc->pGeo->mGeoPath.mPointList.size()>0){
149       glLineWidth(5);
150       glPointSize(5);
151       glDisable( GL_LIGHTING );
152       glColor3f(1,0,0);
153       glBegin( GL_LINE_STRIP );
154       int counter=0;
155       FOR_EACH_LPOINT(pDoc->pGeo->mGeoPath.mPointList,itp)
156         counter++;
157         GPointPtr pPoint = (GPointPtr) *itp;
158
159         Pnt3 Pos = pPoint->getVertex1()->Coord()*pPoint->getCoord()
              + pPoint->getVertex2()->Coord()*(1-pPoint->getCoord());
160
161         glVertex3d(Pos.v[0],Pos.v[1],Pos.v[2] );
162         if(counter>nMaxCountPath)
163           break;
164         GFacePtr pFace = pPoint->getCurFace();
165
166         vector<Pnt3>& SubPointVector = pPoint->getSubPointList();
167         Pnt3& v0 = pPoint->getVertex1()->Coord();
168         Pnt3& v1 = pPoint->getVertex2()->Coord();
169         GVertexPtr pLastVert = pFace->getVertex(
                      *pPoint->getVertex1(), *pPoint->getVertex2());
```

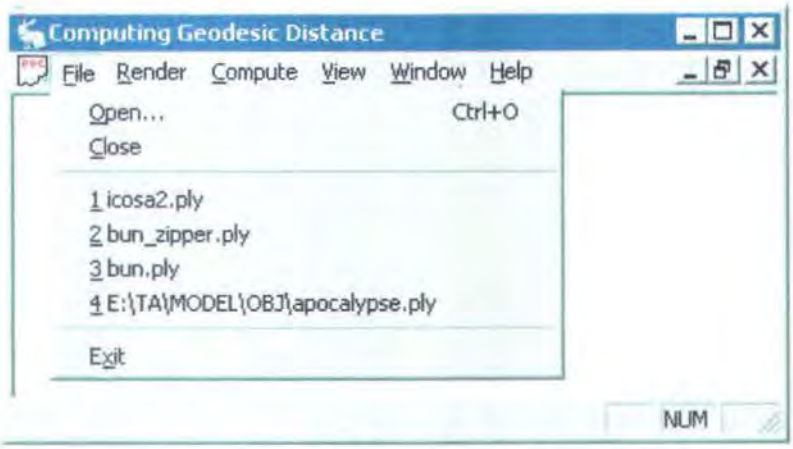
```
170          Pnt3& v2 = pLastVert->Coord();
171          vector<Pnt3>::iterator it;
172          FOR_EACH(it,SubPointVector)
173             Pnt3& coord = *it;
174             Pos = v0*coord.v[0] + v1*coord.v[1] + v2*coord.v[2];
175             glVertex3d( Pos.v[0],Pos.v[1],Pos.v[2]);
176          END
177       END
178       glEnd();
179       glEnable(GL_LIGHTING);
180    }
181 }
```

Gambar 4-34 Potongan Kode Fungsi render pada kelas CGeoDistView

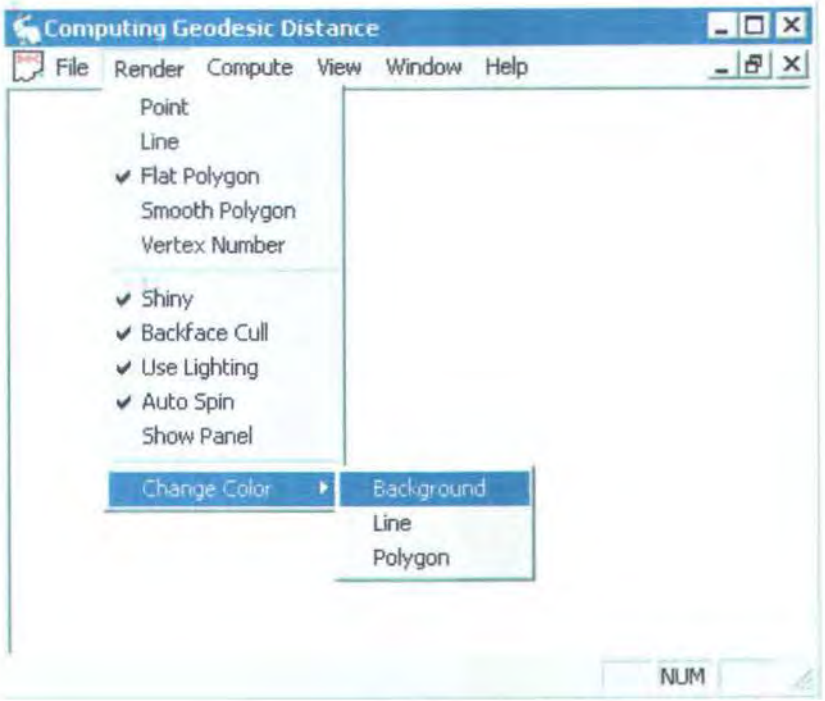
Setelah semua selesai digambar atau fungsi `render` selesai dilakukan, buffer ditampilkan pada kelas `CGeoDistView` dengan memanggil fungsi `SwapBuffers`. Fungsi ini bertujuan untuk menukar buffer yang aktif dengan buffer yang baru.

4.4. IMPLEMENTASI ANTAR MUKA

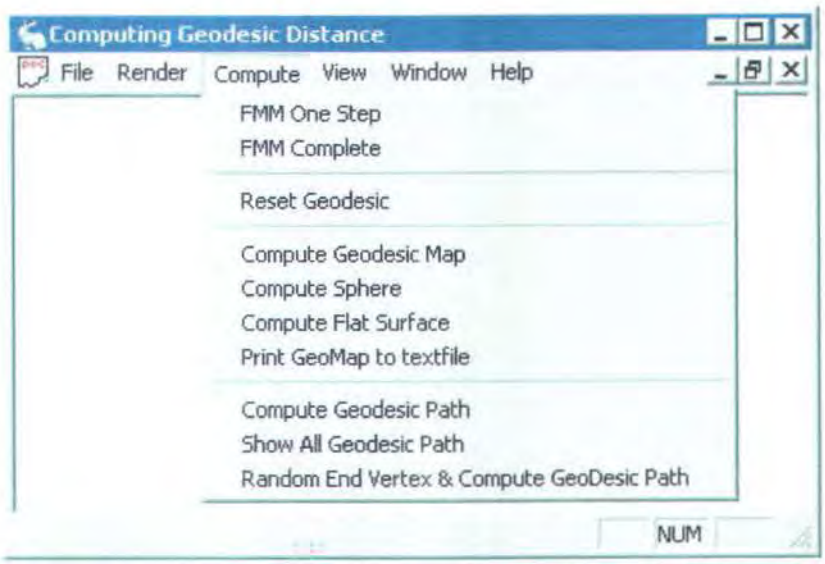
Berikut ini adalah implementasi dari rancangan antar-muka yang telah dibuat pada bab III .



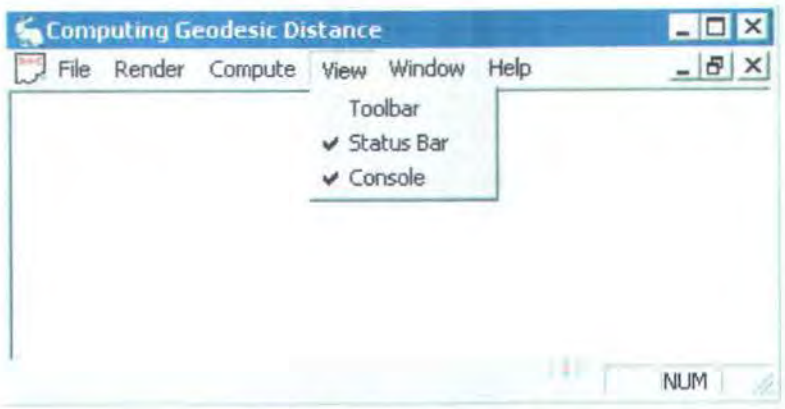
Gambar 4-35 Menu dropdown File



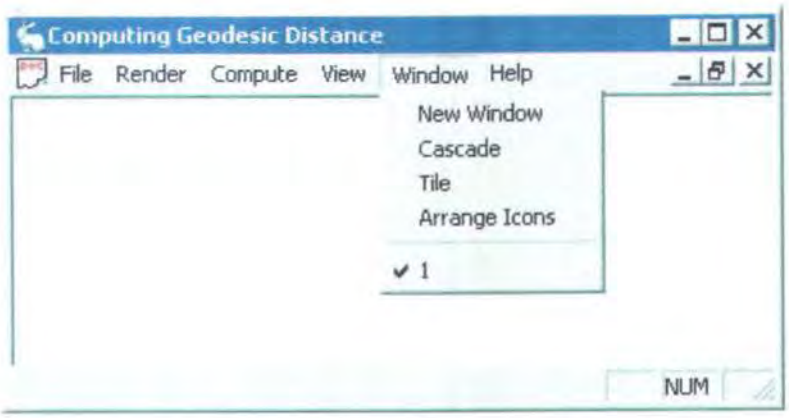
Gambar 4-36 Menu dropdown Render



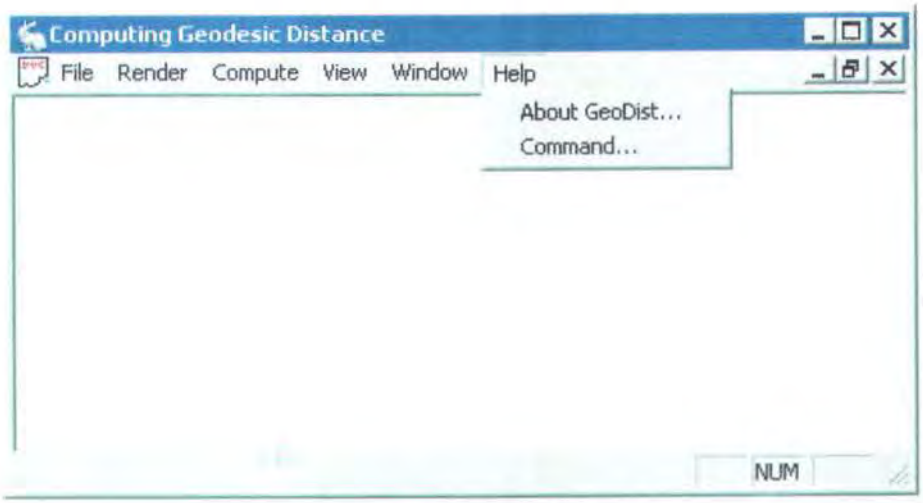
Gambar 4-37 Menu dropdown Compute



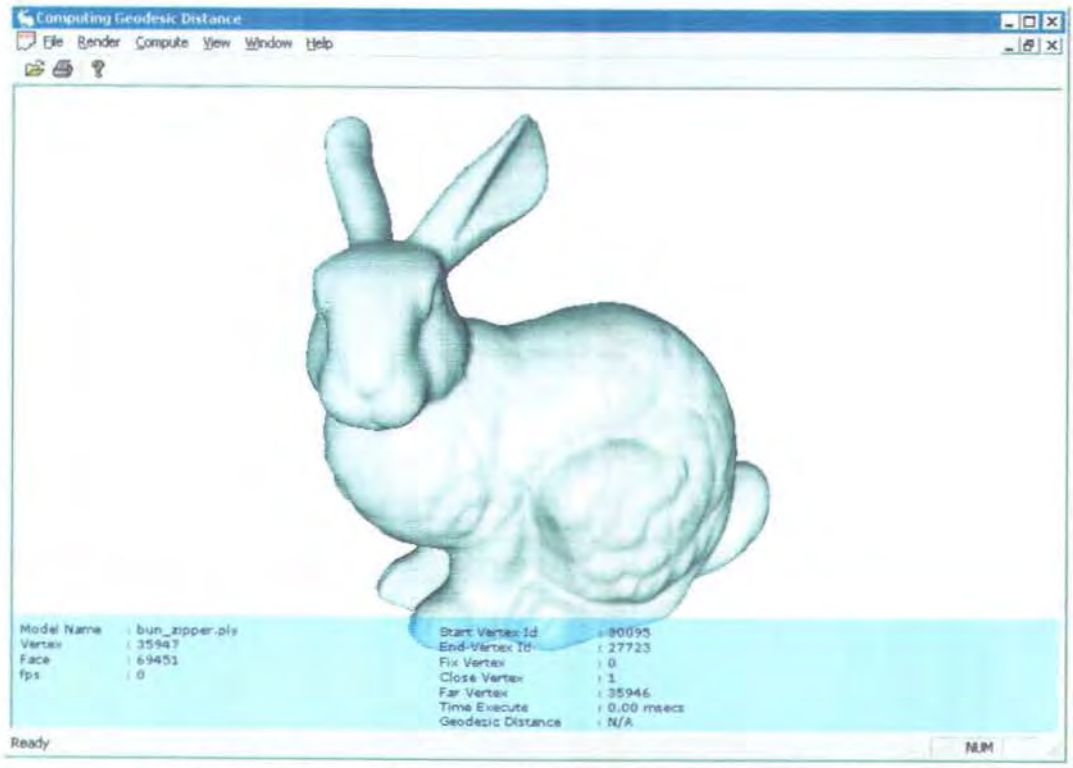
Gambar 4-38 Menu dropdown Window



Gambar 4-39 Menu dropdown Window



Gambar 4-40 Menu dropdown Help



Gambar 4-41 Contoh Model yang dibuka dengan aplikasi yang dibangun.

BAB V
UJI COBA DAN EVALUASI

Pada bab ini dibahas mengenai uji coba terhadap perangkat lunak yang telah dibangun. Uji coba ini dilakukan terhadap data *triangular mesh* dari berkas ply dimana data tersebut dihasilkan dari proses *3D Scanning* yang di-*release* oleh *Stanford University* maupun data dari format file lain yang telah dikonversi ke format file ply.

5.1. Lingkungan Pelaksanaan Uji Coba

Pada sub bab ini akan dijelaskan mengenai lingkungan uji coba aplikasi Perhitungan *Geodesic Distance* dan Pembuatan *Geodesic Path* yang meliputi perangkat keras dan perangkat lunak yang digunakan. Spesifikasi perangkat keras dan perangkat lunak yang digunakan dalam pengujian ini dapat dilihat pada tabel berikut:

Tabel 5-1 Lingkungan Pengujian Aplikasi

Perangkat Keras	Prosesor	: AMD Duron 1100 MHz
	Memory	: 512 MB
	VGA Card	: GeForce 4 MX 440 64 MB
Perangkat Lunak	Sistem Operasi	: Microsoft Windows 2000 Pro.
	Graphic Library	: OpenGL version 1.4

5.2. DATA UJI COBA

Data masukan yang akan digunakan dalam uji coba ini dapat dilihat pada tabel di bawah ini :

Tabel 5-2 Data Model yang dipakai dalam Uji Coba

Data Model		Properti	
		Vertex	Face
1.	sphere.ply	12	20
2.	sphere1.ply	42	80
3.	sphere2.ply	162	320
4.	sphere3.ply	642	1280
5.	sphere4.ply	2562	5120
6.	sphere5.ply	10242	20480
7.	sphere6.ply	40962	81920
8.	sphere7.ply	163842	327680
9.	flat.ply	9	8
10.	flat1.ply	25	32
11.	flat2.ply	81	128
12.	flat3.ply	289	512
13.	flat4.ply	1089	2048
14.	flat5.ply	4225	8192
15.	flat6.ply	16641	32768
16.	flat7.ply	66049	131072
17.	manekin.ply	2732	5420
18.	dragon.ply	5352	10920
19.	bunny.ply	17460	34725
20.	foot.ply	25845	51690
21.	hand.ply	38219	76438
22.	horse.ply	48478	96952
23.	buddha.ply	59958	119920
24.	shoe.ply	78239	156474
25.	topo.ply	88596	175943
26.	mountain.ply	152709	303832
27.	cylinder.ply	853	1752
28.	swissroll.ply	4686	9024
29.	s-surface.ply	3304	6318
30.	limas.ply	8134	16384

5.3. Asumsi Dalam Uji Coba

Dalam pelaksanaan uji coba ini, terdapat beberapa asumsi yang digunakan agar pelaksanaan uji coba dapat berjalan dengan lancar. Asumsi-asumsi tersebut diantaranya :

1. Data masukan yang berfungsi sebagai *inisial Mesh* merupakan *polygonal mesh* dengan bentuk poligon segitiga (*triangular mesh*), setiap *face* hanya mengandung 3 *vertex*.
2. Arah dari urutan *vertex* pada *face* berlawanan arah dengan putaran jarum jam (*counterclockwise*).
3. Setiap *edge* pada *inisial mesh* hanya digunakan oleh maksimal 2 *face* (*Manifold surfaces*).
4. Di dalam data masukan tidak boleh ada *vertex* yang tidak mempunyai *face*. Setiap *vertex* minimal dipakai untuk membentuk sebuah *face*.
5. Di dalam data masukan setiap *vertex* harus dapat dihubungkan ke semua *vertex* lain melalui *edge-edge* yang ada baik secara langsung maupun tak langsung.

5.4. Pelaksanaan Uji Coba dan Evaluasi

Pada sub bab ini akan dijelaskan mengenai uji coba yang dilakukan terhadap perangkat lunak yang telah dibuat. Uji coba tersebut diantaranya :

5.4.1 Uji Coba Kebenaran

Uji coba kebenaran dilakukan untuk menguji validitas dari *geodesic distance* yang dihitung menggunakan algoritme *Fast Marching Method on Triangulated Domain* ini. Percobaan yang dilakukan adalah melakukan proses FMM on TD untuk menghitung *geodesic distance* dari sebuah titik awal ke semua titik lainnya *(single source shortest path problem).* Evaluasi yang dilakukan terhadap hasil uji coba ini meliputi perhitungan rata-rata kesalahan (*error rate*). Rumus perhitungan tingkat kesalahan adalah sebagai berikut :

$$err = \frac{abs(A-B)}{B} \times 100\% \qquad 5.1$$

Dimana :

err = tingkat kesalahan.

A = nilai *geodesic distance* hasil dari algoritme FMM on TD

B = nilai *geodesic distance* sebenarnya

Nilai *err* di atas dihitung pada setiap titik, dimana nilai *geodesic distance* di tiap titik menunjukkan jarak titik tersebut terhadap titik awal. Nilai rata-rata kesalahan *(error rate)* dihitung dengan menghitung nilai rata-rata (mean) dari semua nilai *err* yang dimiliki oleh setiap titik. Nilai *error rate* dirumuskan sebagai berikut :

$$ErrorRate = \frac{\sum_{i=0}^{n-1} err_i}{n} \qquad 5.2$$

Dimana *n* adalah jumlah titik, dan *i* menunjukkan nomor indeks dari tiap titik.

Pelaksanaan uji coba kebenaran ini dibagi menjadi 2 macam yaitu :

5.4.1.1 Uji Coba dengan Sebuah File Model, namun dengan Inputan Titik Awal yang Berbeda

Uji coba ini menggunakan masukan 2 buah model yaitu model permukaan bidang datar (*flat surface*) dan permukaan bola (*sphere*). Alasan penggunaan model-model tersebut dikarenakan pada model tersebut nilai *geodesic distance* sebenarnya dapat dihitung secara geometri analitis. Pada model permukaan bidang datar, nilai *geodesic distance* sebenarnya dapat dihitung dengan menggunakan rumus jarak Euclidian (persamaan 2.1). Sedangkan pada model permukaan bola, nilai *geodesic distance* sebenarnya dapat dihitung dengan menggunakan rumus panjang busur lingkaran (persamaan 2.2).

Pada tiap model, percobaan dilakukan sebanyak 10 kali dengan inputan titik awal yang berbeda-beda ditentukan secara acak (*random*). Hasil percobaan dapat dilihat pada tabel di bawah ini :

Tabel 5-3 Hasil Uji Coba pada Model Permukaan Bidang Datar

Nama File : flat7.ply JumlahTitik : 66049 Jumlah Segitiga : 131072		
Percobaan ke-	**ID Titik Awal**	**Error Rate**
1	0	0,64125%
2	6334	0,57602%
3	19169	0,66612%
4	11478	0,64601%
5	44265	0,54744%
6	59212	0,51233%
7	23281	0,43231%
8	491	0,21712%
9	32391	0,67594%
10	60722	0,65739%
	Rata-rata	0,557193%

Tabel 5-4 Hasil Uji Coba pada Model Permukaan Bola / Sphere

Nama File : sphere6.ply Jumlah Titik : 40962 Jumlah Segitiga : 81920		
Percobaan ke-	**ID Titik Awal**	**Error Rate**
1	0	1,3704%
2	15213	0,57619%
3	33297	0,54759%
4	23796	0,54018%
5	4769	0,6604%
6	10805	0,73596%
7	40933	0,5437%
8	9961	1,0653%
9	292	0,51664%
10	26590	0,53485%
	Rata-rata	0,709121%

5.4.1.2 Uji Coba dengan Bentuk Masukan yang Sama, namun dengan Jumlah Segitiga yang Berbeda

Pada sub bab ini akan dilakukan uji coba kebenaran hasil perhitungan *geodesic distance* dengan menggunakan bentuk model yang sama, namun masing-masing memiliki jumlah segitiga yang berbeda. Uji coba ini dilakukan terhadap bentuk model permukaan bidang datar dan permukaan bola. Untuk tiap bentuk model dibuat 8 file model yang memiliki resolusi yang berbeda. Semakin tinggi resolusi maka semakin banyak jumlah titik/vertex dan segitiga/face yang terdapat pada model tersebut.

Pada tiap model dilakukan sekali percobaan, kemudian dihitung *error rate*-nya. Hasil percobaan pada bentuk permukaan bidang datar dapat dilihat pada tabel berikut :

Tabel 5-5 Hasil Uji Coba pada Bentuk Permukaan Bidang Datar dengan berbagai resolusi

Nama File	Jumlah Titik	Jumlah Segitiga	Error Rate
flat.ply	9	8	7,9203%
flat1.ply	25	32	7,5089%
flat2.ply	81	128	6,058%
flat3.ply	289	512	4,3407%
flat4.ply	1089	2048	2,8702%
flat5.ply	4225	8192	1,7988%
flat6.ply	16641	32768	1,0885%
flat7.ply	66049	131072	0,6413%
		Rata-rata	4,0283375%

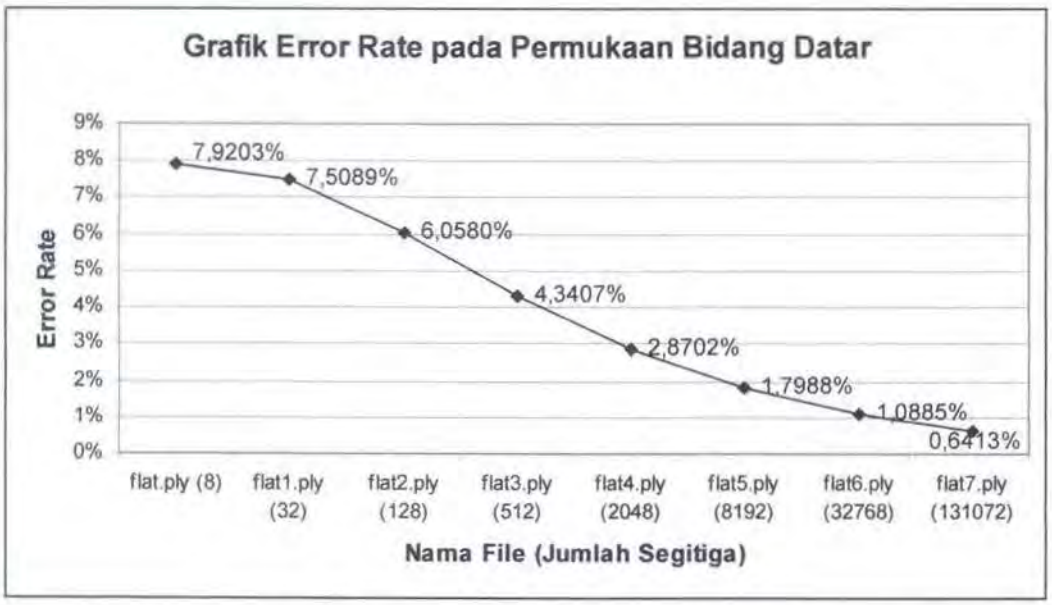
Sedangkan hasil percobaan pada bentuk permukaan bola dapat dilihat pada tabel berikut :

Tabel 5-6 Hasil Uji Coba pada Bentuk Permukaan Bola dengan berbagai resolusi

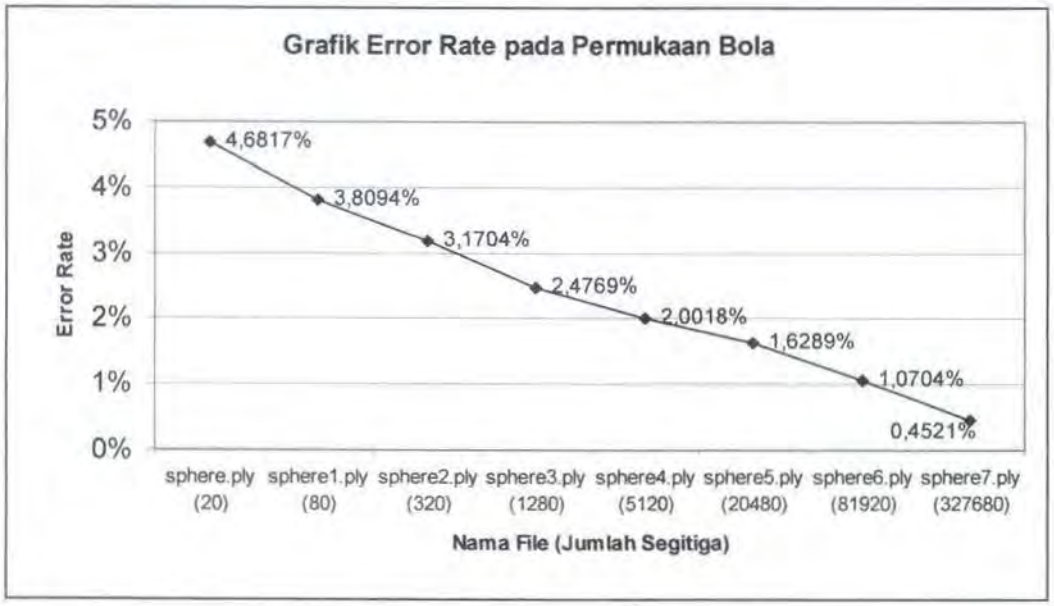
Nama File	Jumlah Titik	Jumlah Segitiga	Error Rate
sphere.ply	12	20	4,6817%
sphere1.ply	42	80	3,8094%
sphere2.ply	162	320	3,1704%
sphere3.ply	642	1280	2,4769%
sphere4.ply	2562	5120	2,0018%
sphere5.ply	10242	20480	1,6289%
sphere6.ply	40962	81920	1,0704%
sphere7.ply	163842	327680	0,4521%
		Rata-rata	2,41145%

Dari hasil percobaan diatas terlihat bahwa banyaknya jumlah segitiga pada suatu model berpengaruh pada tingkat keakuratan nilai *geodesic distance* yang dihasilkan. Semakin banyak jumlah segitiga semakin kecil *error rate* sehingga makin tinggi tingkat keakuratannya.

Gambar 5-1 dan 5-2 menunjukkan grafik error rate terhadap jumlah segitiga yang ada dalam model.



Gambar 5-1 Grafik Error Rate pada Model Permukaan Bidang Datar



Gambar 5-2 Grafik Error Rate pada Model Permukaan Bola

5.4.2 Uji Coba Kecepatan Eksekusi Proses Fast Marching Method on Triangulated Domain

Pada sub bab ini akan dilakukan uji coba untuk mengetahui waktu yang dibutuhkan perangkat lunak untuk melakukan proses Fast Marching Method.

Percobaan yang dilakukan adalah melakukan proses FMM on TD untuk menghitung *geodesic distance* dari sebuah titik awal ke semua titik lainnya *(single source shortest path problem).* Evaluasi yang dilakukan terhadap hasil uji coba ini adalah perhitungan rata-rata kecepatan eksekusi.

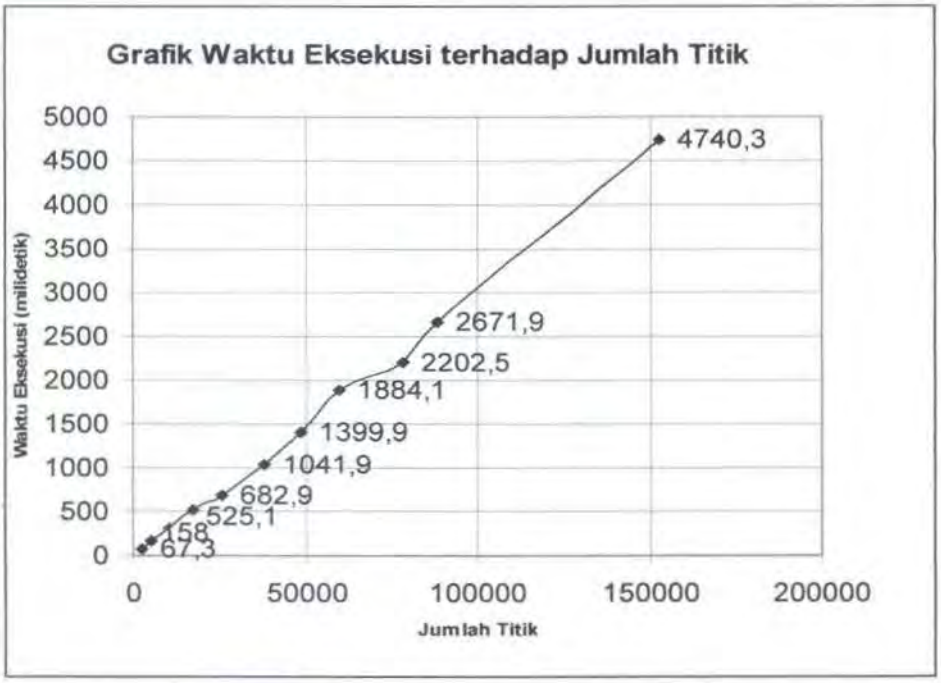
Uji coba ini dilakukan dengan menggunakan 10 file model yang berbeda-beda resolusinya, diurutkan dari resolusi rendah ke resolusi tinggi. Pada tiap model, percobaan dilakukan sebanyak 10 kali dengan inputan titik awal yang berbeda-beda ditentukan secara acak (*random*), kemudian dicatat waktu yang dibutuhkan untuk sekali eksekusi dan dihitung rata-ratanya. Hasil percobaan dapat dilihat pada tabel di bawah ini :

Tabel 5-7 Hasil Uji Coba pada berbagai file model dengan berbagai resolusi

Nama File	Jumlah Titik	Waktu Eksekusi tiap Percobaan (dlm milidetik)										Rata-rata
		1	2	3	4	5	6	7	8	9	10	
manekin.ply	2732	62	62	63	78	63	78	79	63	62	63	67,3
dragon.ply	5352	172	157	156	156	157	157	156	156	157	156	158
bunny.ply	17460	532	531	516	532	531	515	531	531	516	516	525,1
foot.ply	25845	672	688	687	704	656	672	687	688	688	687	682,9
hand.ply	38219	1015	1046	1031	1063	1047	1015	1031	1046	1062	1063	1041,9
horse.ply	48478	1406	1375	1422	1390	1391	1422	1406	1359	1422	1406	1399,9
buddha.ply	59958	1844	1828	1875	1859	1984	1875	1860	1950	1907	1859	1884,1
shoe.ply	78239	2140	2172	2219	2140	2261	2265	2203	2219	2187	2219	2202,5
topo.ply	88596	2578	2516	2500	2703	2641	2766	2985	2656	2640	2734	2671,9
mountain.ply	152709	4719	4750	4532	4812	4641	4781	4656	4985	4777	4750	4740,3

Dari proses uji coba yang telah dilakukan didapatkan hasil yang digambarkan dalam bentuk grafik. Pada gambar 5-3 ditampilkan grafik perbandingan waktu eksekusi terhadap jumlah titik yang ada pada model. Dari grafik tersebut dilihat bahwa semakin banyak jumlah titik, maka waktu yang diperlukan semakin lama. Sebagai perbandingan, dapat dilihat pada gambar 5-4

yang menunjukkan grafik kompleksitas algoritme FMM $O(n \lg n)$ seperti yang telah dijelaskan pada sub bab 2.2.4.

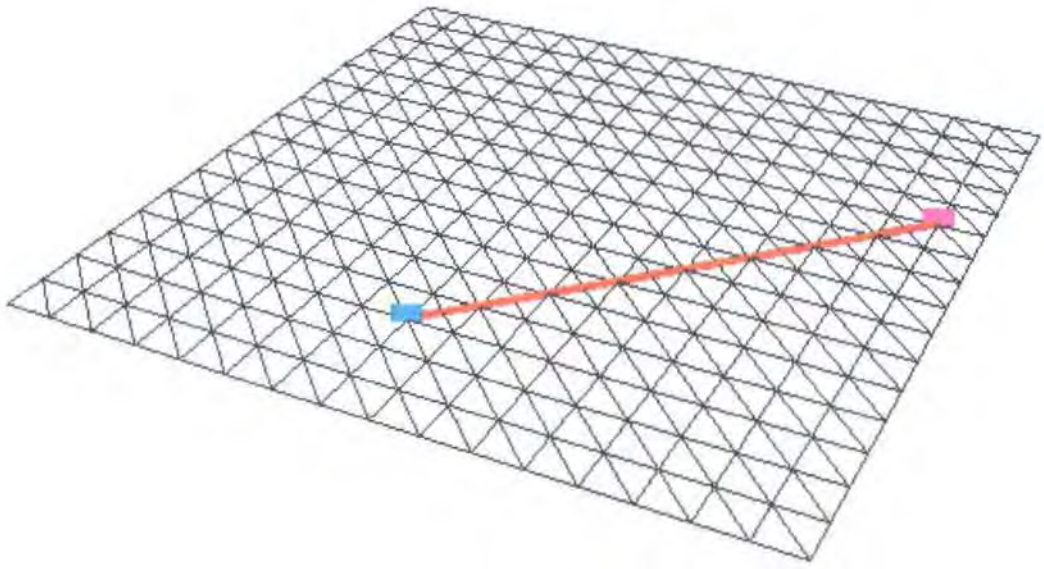


Gambar 5-3 Grafik Waktu Eksekusi terhadap Jumlah Titik

Gambar 5-4 Grafik Kompleksitas Algoritme FMM O(n log n)

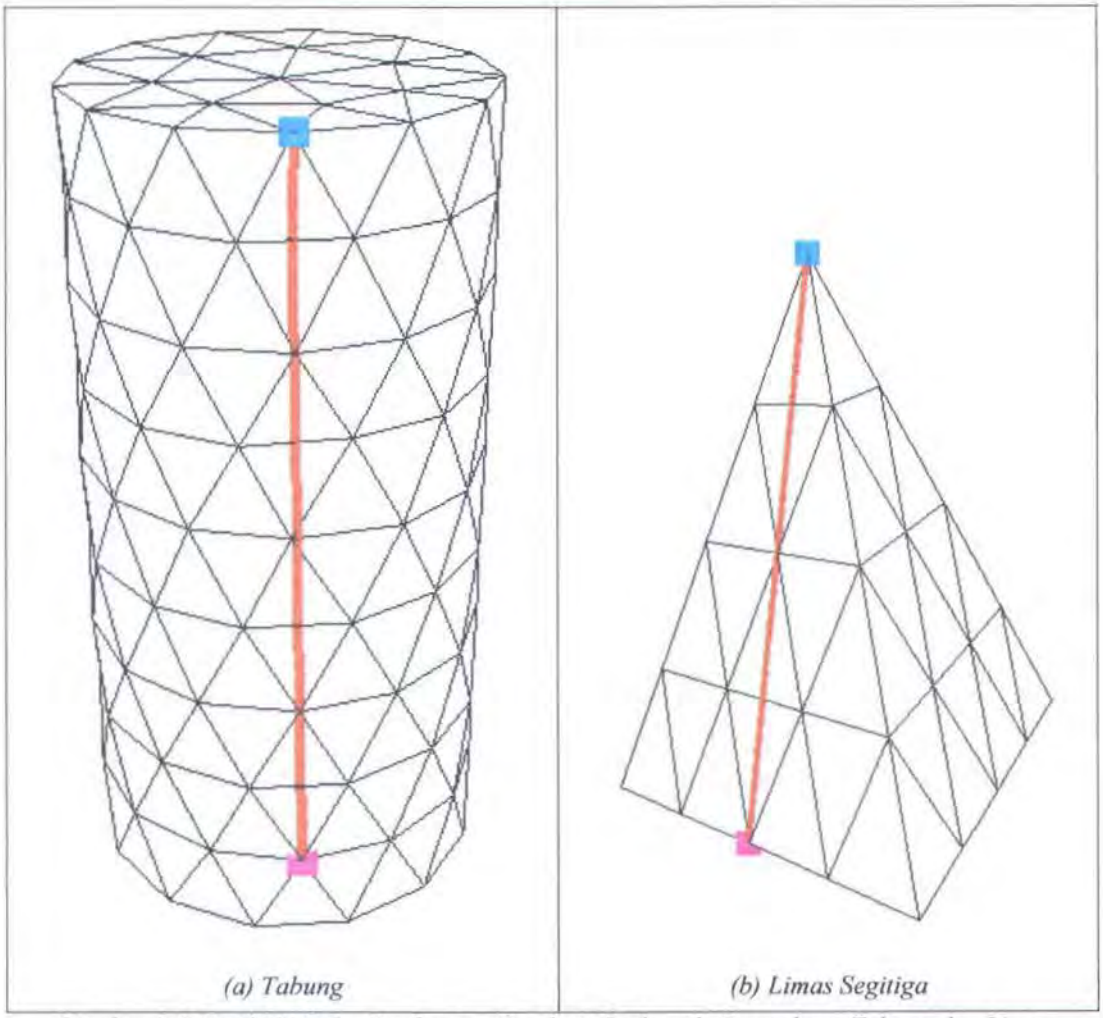
5.4.3 Uji Coba Visualisasi Geodesic Path

Pada uji coba ini dilakukan proses pembuatan *geodesic path* dan kemudian divisualisasikan pada model. Seperti yang telah dijelaskan pada sub bab 2.1 bahwa *geodesic path* adalah lintasan pada permukaan objek 3 dimensi yang merepresentasikan *geodesic distance*. Sehingga *geodesic path* yang merepresentasikan *geodesic distance* dari 2 titik pada permukaan bidang datar adalah sebuah garis lurus yang menghubungkan kedua titik tersebut. Untuk itu dilakukan uji coba pembuatan *geodesic path* pada model permukaan bidang datar. Pada gambar 5-5 tampak hasil uji coba pembuatan *geodesic path* pada permukaan bidang datar adalah garis lurus yang menghubungkan titik awal dan titik akhir.



Gambar 5-5 Hasil Uji Coba Pembuatan Geodesic Path pada Permukaan Bidang Datar

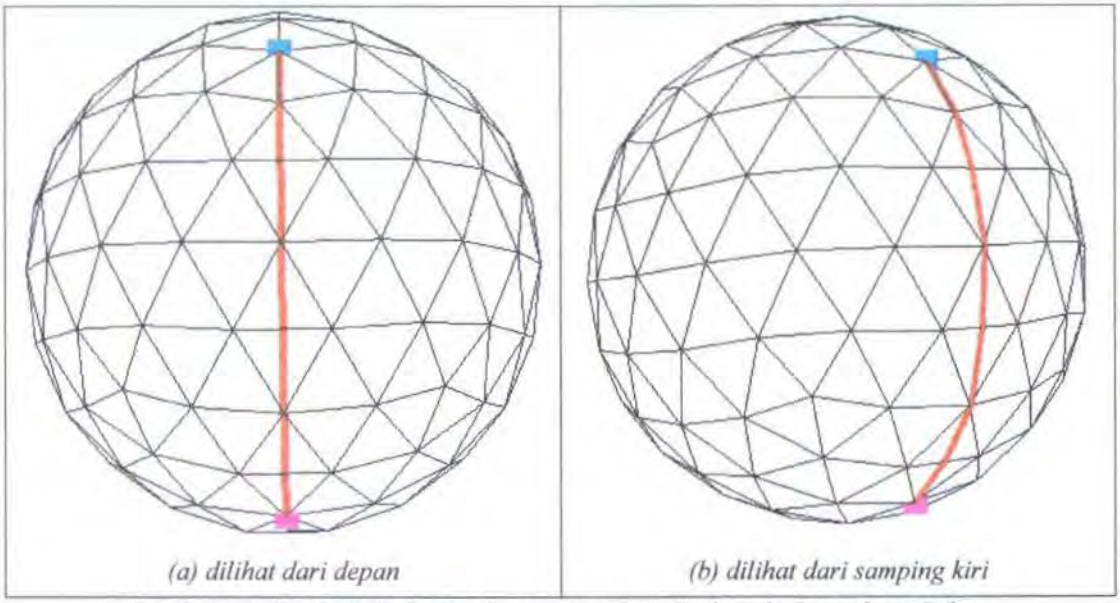
Selain pada permukaan bidang datar, dilakukan juga uji coba pada model permukaan bangun ruang sederhana seperti silinder dan limas. Pada gambar 5-6 tampak *geodesic path* yang berupa garis lurus pada permukaan silinder dan limas yang menghubungkan titik awal dan titik akhir



(a) Tabung *(b) Limas Segitiga*

Gambar 5-6 Hasil Uji Coba Pembuatan Geodesic Path pada Permukaan Tabung dan Limas

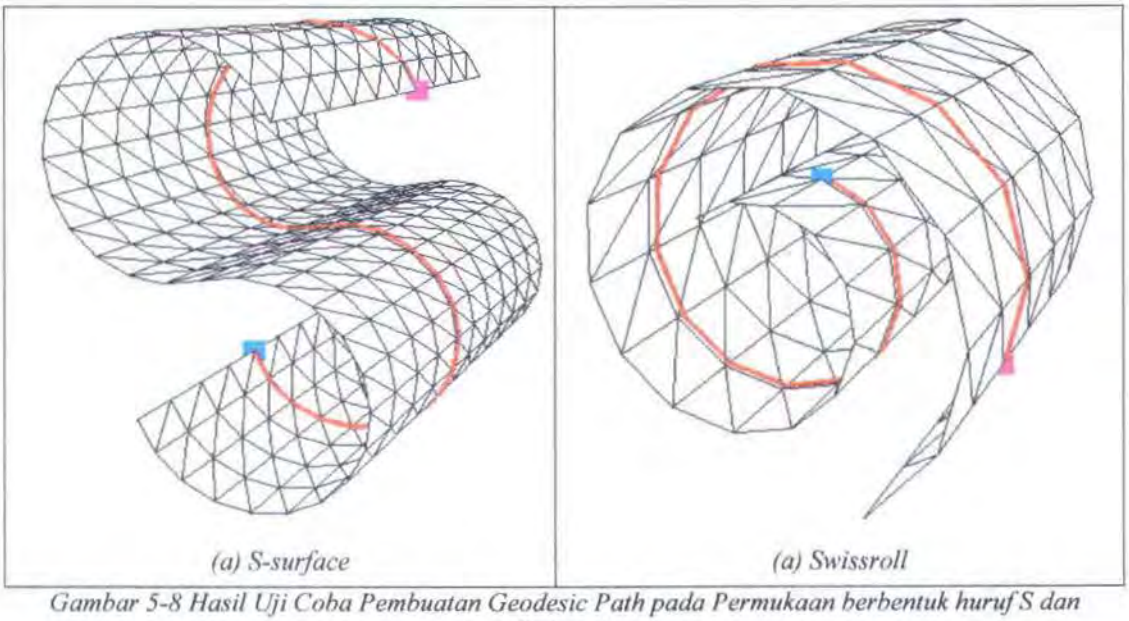
Pada sub bab 2.1 juga telah dijelaskan bahwa *geodesic path* pada permukaan bola (*sphere*) adalah berupa busur lingkaran pada permukaan. Untuk itu dilakukan uji coba pembuatan *geodesic path* pada sebuah bola. Pada Gambar 5-7 tampak *geodesic path* pada permukaan sphere. Pada gambar sebelah kiri, geodesic path tampak seperti garis lurus karena sphere dilihat dari depan. Pada gambar sebelah kanan sphere dilihat dari pinggir.



(a) dilihat dari depan *(b) dilihat dari samping kiri*

Gambar 5-7 Hasil Uji Coba Pembuatan Geodesic Path pada Permukaan Sphere

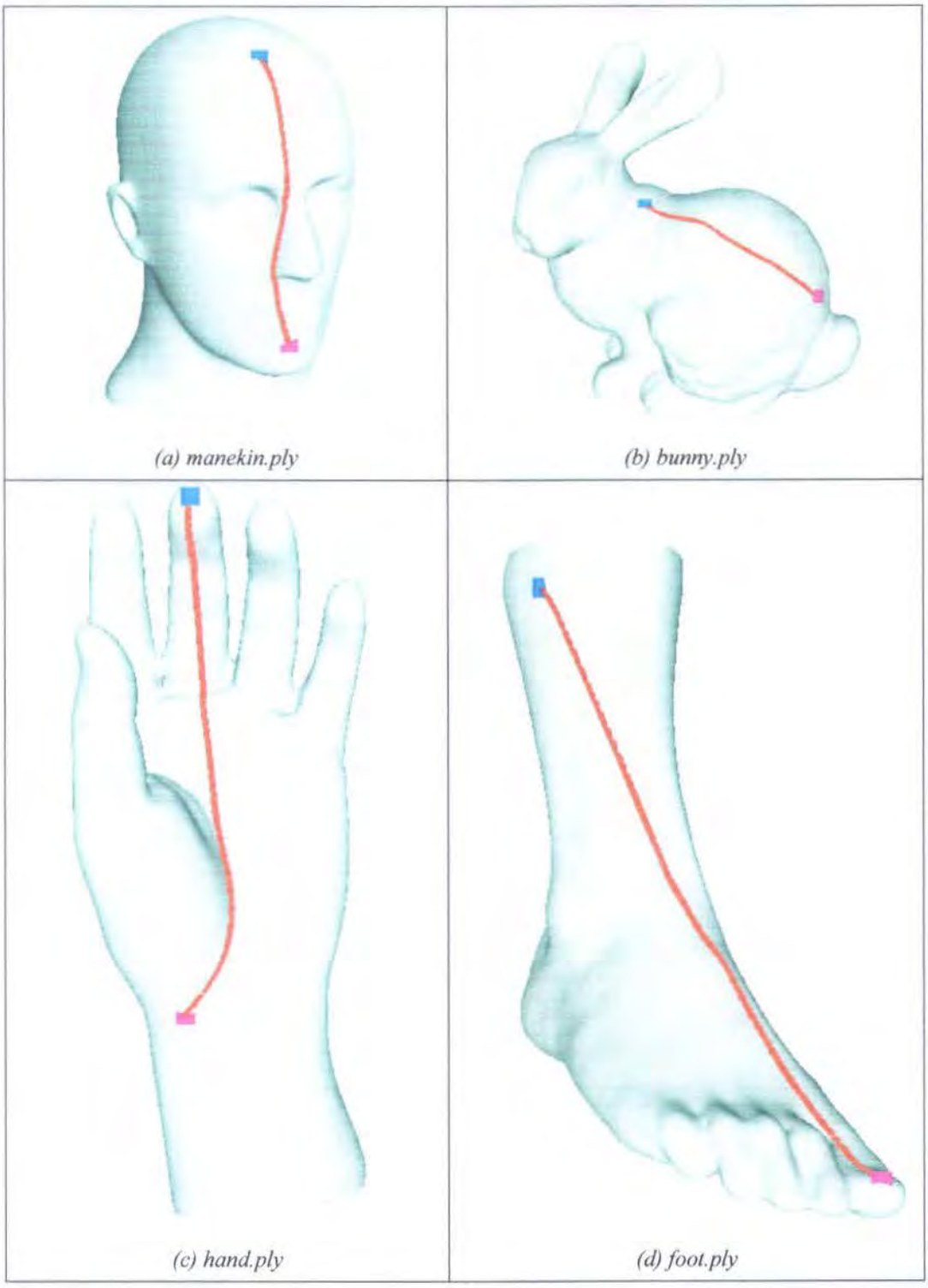
Pada model permukaan lain seperti permukaan berbentuk huruf S dan permukaan swissroll (gulungan spiral). *Geodesic path* akan berbentuk sesuai dengan bentuk permukaannya. Pada gambar 5-8 tampak hasil uji coba visualisasi geodesic path pada permukaan huruf S dan permukaan swissroll



(a) S-surface *(a) Swissroll*

Gambar 5-8 Hasil Uji Coba Pembuatan Geodesic Path pada Permukaan berbentuk huruf S dan Swissroll

Berikut gambar-gambar hasil uji coba perangkat lunak untuk membentuk geodesic path pada model-model lain yang lebih rumit yaitu model-model hasil pemindaian 3 dimensi dan model *terrain* (permukaan kulit bumi).

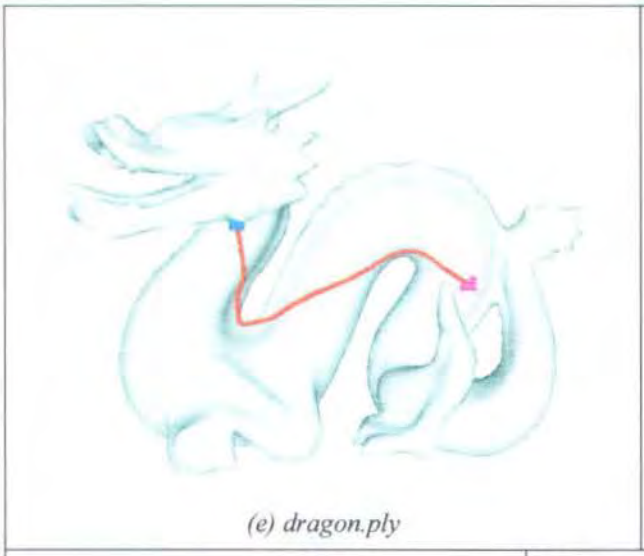


(a) manekin.ply

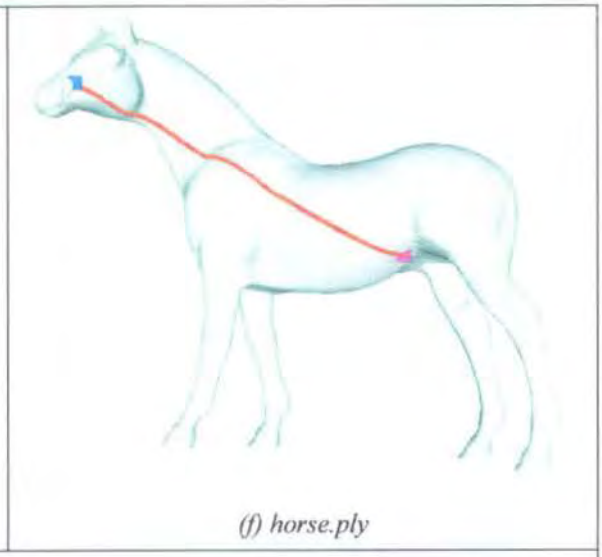
(b) bunny.ply

(c) hand.ply

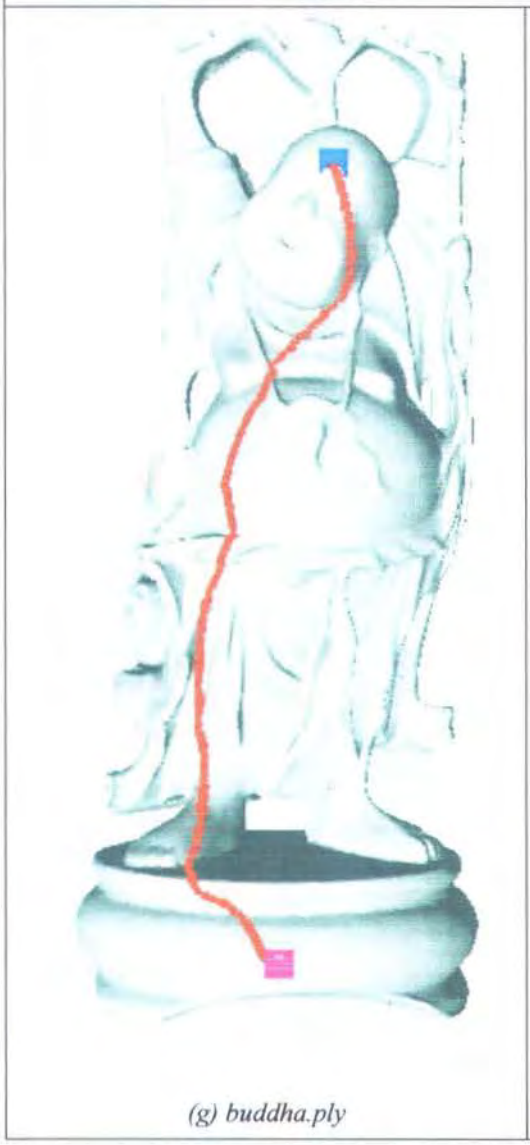
(d) foot.ply



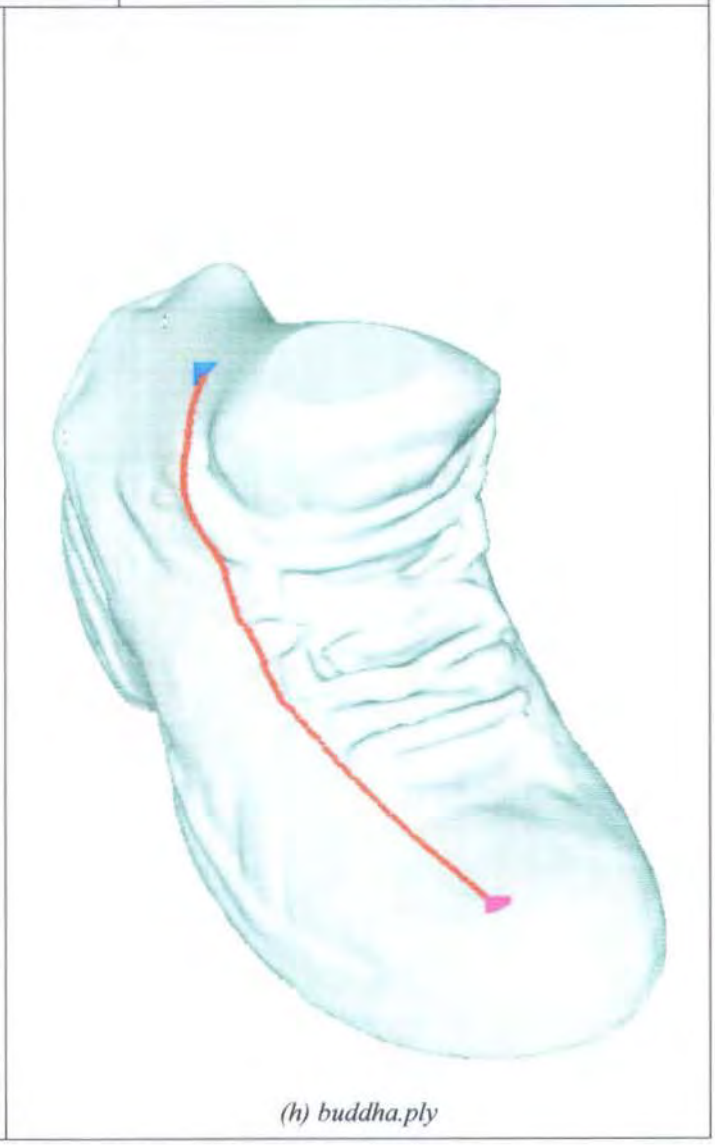
(e) dragon.ply



(f) horse.ply

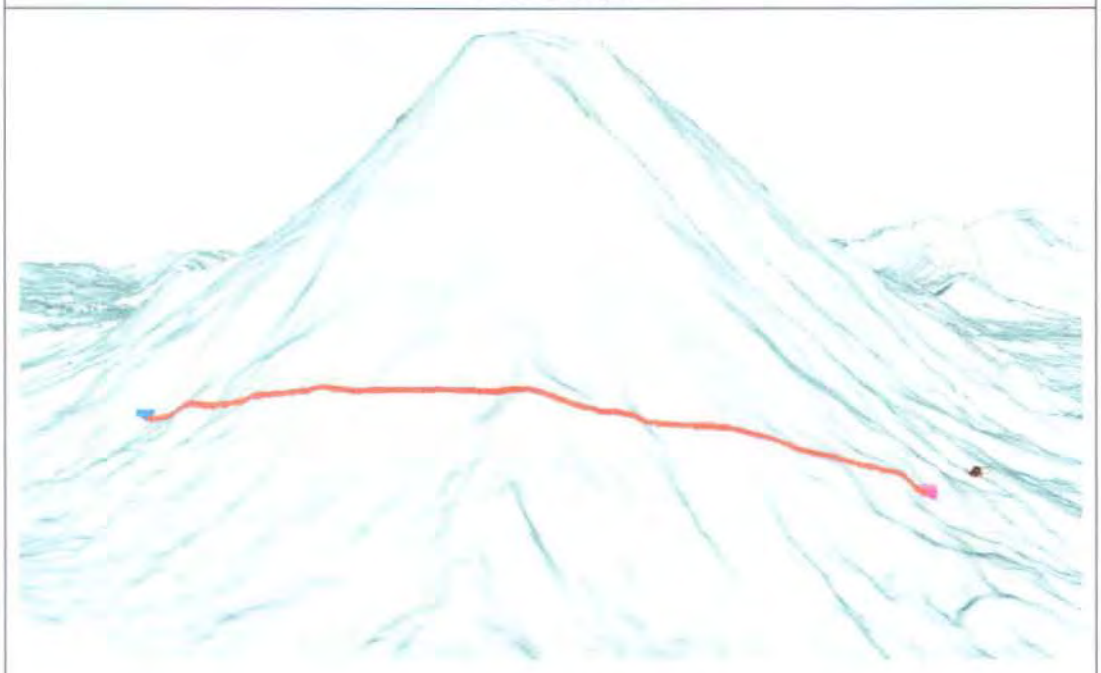


(g) buddha.ply



(h) buddha.ply

(i) topografi.ply



(j) mountain.ply

Gambar 5-9 Hasil Uji Coba Pembuatan Geodesic Path pada Berbagai Model yang Rumit

BAB VI
KESIMPULAN DAN SARAN

Bab ini berisi kesimpulan yang diperoleh berdasarkan uji coba yang telah dilakukan. Selanjutnya diberikan beberapa saran yang mungkin dapat digunakan untuk mengembangkan hasil yang diperoleh pada tugas akhir ini.

6.1. KESIMPULAN

Setelah dilakukan serangkaian uji coba dan analisis terhadap perangkat lunak yang dibuat, maka dapat diambil kesimpulan sebagai berikut:

1. Algoritme *Fast Marching Method on Triangulated Domain* dapat digunakan sebagai pendekatan untuk menghitung *geodesic distance* antara 2 titik pada permukaan objek *triangular mesh* dengan tingkat keakuratan lebih dari 95 %.
2. Kecepatan eksekusi algoritme FMM on TD dipengaruhi oleh jumlah titik. Kompleksitas waktunya adalah $O(n \lg n)$, dimana n adalah jumlah titik yang dimiliki oleh model. Jadi semakin banyak titik yang ada maka semakin lama waktu yang dibutuhkan untuk melakukan perhitungan.
3. Banyaknya segitiga yang membentuk *triangular mesh* berpengaruh pada *error rate* dari hasil algoritme FMM on TD. Semakin banyak segitiga yang ada semakin sedikit *error rate* yang dihasilkan. Banyaknya segitiga juga berpengaruh pada jumlah titik pada *mesh*. Semakin banyak segitiga yang ada semakin banyak jumlah titik yang ada

6.2. SARAN

Perangkat lunak perhitungan *geodesic distance* dan pembuatan *geodesic path* pada objek *triangular mesh* ini dirancang dengan menggunakan pendekatan berorientasi objek. Perangkat lunak ini dapat membuka bermacam-macam tipe format data masukan. Pada tugas akhir ini tipe model yang diimplementasikan hanya tipe Poligon Model dengan bentuk segitiga, dan *parser* yang digunakan untuk membaca data masukan hanya untuk format data ply (*PlyParser*). Beberapa kemungkinan pengembangan lebih lanjut dari perangkat lunak ini adalah sebagai berikut :

1. Agar perangkat lunak dapat digunakan untuk melakukan proses triangulasi pada poligon model dengan bentuk selain segitiga atau model non *triangular mesh*, sehingga meskipun model yang dibuka bukan merupakan *triangular mesh*, tetap dapat dihitung *geodesic distance*-nya dengan menggunakan algoritme *Fast Marching Method on Triangulated Domain*.
2. Agar perangkat lunak dapat digunakan untuk membuka tipe model lainnya, seperti *terrain* model (DEM file) dapat dibuatkan kelas model tersendiri selain Poligon Model.
3. Format untuk menyimpan data poligon model sebagai data objek 3 dimensi selain ply ada bermacam-macam seperti VRML, inventor, obj, poly. Untuk dapat membaca data tersebut dapat dibuatkan *parser* untuk masing masing format (*VRMLParser, ObjParser, InventorParser atau PolyParser*).

4. Beberapa kelas yang digunakan dalam perangkat lunak ini dapat dirubah ke dalam bentuk sistem *Plug-in* (*dll library*) sehingga dapat memudahkan untuk dipakai dalam pengembangan aplikasi lain seperti aplikasi pendataran permukaan *(surface flattening)*, aplikasi *model remeshing*, dan lain-lain.

DAFTAR PUSTAKA

[1] Sethian J. A.: *Level Set Methods and Fast Marching Methods*, 2nd ed, Cambridge University Press, 1999

[2] Sethian J. A.: *Fast Marching Methods*, SIAM Review, no. 41, July 1999.

[3] Sethian J. A.: *A Fast Marching Level Set Method for Monotonically Advancing Fronts*. Proceedings of National Academy of Sciences, vol. 93, no. 4, pp. 1591-1595, 1996.

[4] Kimmel R. and Sethian J. A.: *Computing Geodesic Paths on Manifolds*. Proc. Natl. Acad. Sci., vol. 95, no. 15, pp. 8431-8435, 1998.

[5] Gil Zigelman, Ron Kimmel, and Nahum Kiryati. "*Texture Mapping Using Surface Flattening via Multidimensional Scaling*". IEEE Transaction on Visualization and Computer Graphics, vol. 8, no 2, April-June 2002.

[6] Adi L. and Kimmel R.: *Interactive Edge Integration on Painted Surfaces*. Computer Science, Technion-Israel Institute of Technology, 2003.

[7] Cohen L.D. and Kimmel R.: *Global Minimum for Active Contour Models: A Minimal Path Approach*. International Journal of Computer Vision, vol. 24, no. 1, pp. 57–78, Aug. 1997.

[8] Peyré G. and Cohen L. D.: *Geodesic Remeshing Using Front Propagation*. Proc. IEEE VLSM, 2003.

[9] Peyré G. and Cohen L. D.: *Geodesic Computations for Fast and Accurate Surface Flattening*. Proc. IEEE VLSM, 2003.

[10] Marcin Novotni and Reinhard Klein : *Computing Geodesic Distances on Triangular Meshes*. Insitut f¨ur Informatik II, Bonn, Germany, 2002.

[11] Dimas Mart´inez, Luiz Velho, and Paulo Cezar Carvalho : *Geodesic Paths on Triangular Meshes.* IMPA–Instituto Nacional de Matem´atica Pura e Aplicada, 2003.

[12] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein : *Introduction to Algorithms*, 2nd Edition, The MIT Press, 2001.

[13] W.H. Press et Al., *Numerical Recipes in C : the art of computer programming*. Cambridge University Press, 1988.

[14] Steven C. Chapra and Raymon P. Canale: *Numerical Methods for Engineers*. Mc Graw Hill Book Company, 1985.

[15] John H. Mathews and Kurtis K. Fink : *Numerical Methods Using Matlab*, 4th Edition, Prentice-Hall Inc. 2004.

DAFTAR PUSTAKA

[1] Adi L. dan Kimmel R. "Interactive Edge Integration on Painted Surfaces". Computer Science, Technion-Israel Institute of Technology, 2003.

[2] Chapra S.C. dan Canale R.P. "Numerical Methods for Engineers". Mc Graw Hill Book Company, 1985.

[3] Cohen L.D. dan Kimmel R. "Global Minimum for Active Contour Models: A Minimal Path Approach". International Journal of Computer Vision, vol. 24, no. 1, pp. 57–78, Aug. 1997.

[4] Cormen T.H. dan Leiserson C.E. "Introduction to Algorithms", 2nd Edition, The MIT Press, 2001.

[5] Kimmel R. dan Sethian J.A. "Computing Geodesic Paths on Manifolds". Proc. Natl. Acad. Sci., vol. 95, no. 15, pp. 8431-8435, 1998.

[6] Mart´inez D., Velho L., dan Carvalho P.C. "Geodesic Paths on Triangular Meshes". IMPA–Instituto Nacional de Matem´atica Pura e Aplicada, 2003.

[7] Mathews J.H. dan Fink K.K. "Numerical Methods Using Matlab", 4th Edition, Prentice-Hall Inc. 2004.

[8] Novotni M. dan Klein R. "Computing Geodesic Distances on Triangular Meshes". Insitut f¨ur Informatik II, Bonn, Germany, 2002.

[9] Peyré G. dan Cohen L.D. "Geodesic Remeshing Using Front Propagation". Proc. IEEE VLSM, 2003.

[10] Peyré G. dan Cohen L.D. "Geodesic Computations for Fast and Accurate Surface Flattening". Proc. IEEE VLSM, 2003.

[11] Press W.H., Teukolsky S.A., Veterling W.T., dan Flannery B.P. "Numerical Recepies in C". Cambridge University Press, 1992.

[12] Sethian J.A. "Level Set Methods and Fast Marching Methods", 2nd ed, Cambridge University Press, 1999

[13] Sethian J.A. "Fast Marching Methods", SIAM Review, no. 41, July 1999.

[14] Sethian J.A. "A Fast Marching Level Set Method for Monotonically Advancing Fronts". Proceedings of National Academy of Sciences, vol. 93, no. 4, pp. 1591-1595, 1996.

[15] Zigelman G., Kimmel R., dan Kiryati N. "Texture Mapping Using Surface Flattening via Multidimensional Scaling". IEEE Transaction on Visualization and Computer Graphics, vol. 8, no 2, April-June 2002.